KNOWLEDGE PROGRAMMING FOR COMPUTER-AIDED DESIGN:

AN APPLICATION TO THE DESIGN OF ENERGY INTEGRATION NETWORKS

by

CHETAN MEHTA

B. Tech., Indian Institute of Technology, Bombay, 1982.
M. S., Kansas State University, 1986.

_____

A THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

Approved by:

_____
Co-Major Professor

_____
Co-Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

CHAPTER 1.   INTRODUCTION

Automation of engineering design is one of the most promising areas for applying computer technology. Efforts at automating the engineering design tasks have resulted in two categories of products: design tools which act as aids to a designer and do not have capabilities to invent a new structure on their own, and automated design systems which generate structures essentially on their own. Of these two, the former have enjoyed wide spread acceptance and confidence among design engineers, whereas the latter are not "trusted" by practicing engineers because of their inability to match up with an experienced designer's performance and to justify the solutions they provide. In this respect, engineering design is similar to medical diagnosis; both have stringent requirements of performance and justification. In medical diagnosis, the survival (recovery) of a patient is critical, and in engineering design, the survival of a company or a plant is at stake.

The present work is towards the development of an automated design system by employing an AI based approach. Engineering design process typically consists of four stages: synthesis, analysis, evaluation and optimization. The last three steps, which constitute the detailed design phase, have enjoyed reasonable success at automation. The lack of success of the present day automated design systems can be attributed to ineffective automation of the first stage of the design process, which constitutes the conceptual phase of design. Ability to conceive innovative designs is a hallmark of human intelligence. This fact alone necessitates an AI based approach to its automation, since AI is the

discipline (of computer science) that provides techniques for encapsulating intelligent human behavior. Expert designers acquire the ability to conceive "good" designs by learning through experience. Therefore, to evolve usable automated design systems, it is imperative that attempts be made to characterize, model and quantify the designers' thought processes involved in visualization, conception, and evaluation of new designs. These thought processes include tasks such as decision-making in the presence of incomplete and/or uncertain knowledge, subjective evaluations and qualitative trade-offs, planning with constraints and resource allocation.

Existing approaches to automate the conceptual design phase rely mainly upon the numerical estimation of the operational characteristics of the system being designed for making the structural decisions. The quality of the solution obtained by such an approach depends on the accuracy of the estimation. Since the operational characteristics depend on several parameters, some of which are not known until the detailed design phase, the estimation procedures are usually not very accurate. Furthermore, these procedures tend to be very complex and computation intensive. These factors, coupled with the excessively large number of possible structural configurations, render the present methods too complex and ineffective in solving large industrial problems.

Good designers do not reason on the basis of numerical estimation of the operational characteristics [see, e.g., Williams, 1985]; rather, they use their knowledge about the qualitative relationships between the structural and operational characteristics of the system being designed.

2

An AI based approach to design automation advocates that for developing effective automated design systems, this knowledge should be identified and captured in computer based systems.   This approach has been successfully applied to the problem of digital circuit design [De Kleer, 1985].

The scope of the present work is to use an AI based approach towards the automation of the conceptual phase of design. We present a state-space search formulation of a typical design problem, viz., that of synthesizing energy integration networks for chemical and power plants. We use the available domain knowledge to direct the search for an optimal solution.   To demonstrate the feasibility of our approach, a prototype systems has been implemented in the object-oriented paradigm of LOOPS environment on a Xerox AI workstation. The system is capable of generating a set of network configurations that possess the desired structural characteristics for a given energy integration problem.

The most significant aspect of our approach is that it is not based on the estimation of approximate real cost ($/year) of the candidate structures. So far, all existing methods, used for computer based synthesis of energy integration networks, require the cost computation and provide a single structure that has the minimum estimated real cost. The disadvantage of such an approach is that the candidate structure so generated may not be an overall optimal design; it may possess unacceptable operational characteristics, when analyzed and evaluated during the detailed design phase. Furthermore, several structures have costs that are very close to each other and one can not guarantee any particular structure to have the minimum cost, since the cost estimation

3

function is usually not accurate by more than 20%. These features render the performance of the present methods unacceptable to most industrial designers (see, e.g., Linnhoff et al, 1982; Barton et al, 1987). Instead of the numerical estimation of the cost, the present approach relies upon the domain knowledge in the form of qualitative relationships between the structural characteristics and the cost of the completed structure. The prototype system developed in the present work attempts to generate a set of structures that have acceptably lower costs and provide a designer with several candidates for evaluating the other operational characteristics in the detailed design phase. It, therefore, conserves computational resources by not searching for the minimum-cost structure.

The implementation reported herein constitutes the first prototype in the ongoing HENSYN project in the Department of Chemical Engineering. The project aims to develop an "intelligent" automated design system for energy integration networks. Additionally, it is intended that this prototype will aid the participating knowledge engineers and domain experts in

(a) extraction and formalization of additional knowledge required for the synthesis of energy integration networks,

(b) testing the adequacy and efficiency of new design strategies.

This thesis is divided into five chapters. Chapter 2 presents a background of the domain, the description of the problem, and the domain knowledge employed in the present work. The third chapter presents a brief discussion of search systems, and formulates the synthesis task under consideration as a state-space search problem. We define the

4

state-space of the problem, the operators and control strategy for manipulating the domain knowledge. Chapter 4 deals with the implementation of the proposed state-space search formulation for synthesizing energy integration networks in LOOPS environment. The chapter contains an overview of LOOPS and the LOOPS representation of the three components of the search system for the problem under consideration. Additionally, we describe the workings of the user interface. The chapter ends with an analysis of the performance of the prototype. The last chapter summarizes the accomplishments of the present work and identifies the future enhancements of the prototype.

CHAPTER 2.   HEAT EXCHANGER NETWORK SYNTHESIS

In a chemical plant, a number of streams are required to be heated or cooled, each from one temperature (*source temperature*) to another (*target temperature*). Traditionally, the heating is carried out by steam and cooling by cold water. However, if the energy of "hot" streams can be utilized to heat the "cold" streams, considerable savings can result. Such energy transfer between a pair of streams is carried out using a device called *heat exchanger*. A network of heat exchangers, chosen judiciously, can drastically reduce the amount of utilities (steam and cooling water) required to run a plant. Given the prevailing scales of operation, this can translate into annual savings of millions of dollars for the plant. Consequently, the optimal design of a heat exchanger network (HEN) is a problem of significant interest.

2.1   BACKGROUND

A *stream* in a chemical plant is any material, mainly in liquid and/or gaseous form, flowing through a pipe. The material of a stream is modified or transformed while transiting through a processing unit of the plant. Each transformation results into a change in the energy content of the stream. This change manifests itself in a variety of forms. Of interest to the present work is the change in a specific form of energy, called *enthalpy*, at constant pressure. Such a change in enthalpy changes the temperature of the material. Associated with each

6

stream are several characteristics; of these, relevant to the present work are the following three:

  (1) temperature, denoted by T,

  (2) rate of flow of the material, denoted by m,

  (3) specific heat at constant pressure, denoted by c_p.

The last one, *specific heat*, is a measure of the ease of changing the temperature of a stream. It is expressed as the change in temperature of unit amount of material, for a unit change in the enthalpy of the material at constant pressure.

The most common and obvious way of changing the energy content of a stream is by supplying or removing heat. When Q units of heat is supplied to (or removed from) a stream, it results in a change ΔH in the enthalpy content of the stream. The first law of thermodynamics specifies that under reasonable assumptions, this change in enthalpy is equal to the amount of heat (Q) supplied to or removed from the stream [see, e.g., Kyle, 1983]:

$$Q = \Delta H \tag{2-1}$$

Note that in this relationship, both the quantities are positive if heat is supplied to the stream and both are negative if heat is removed from the stream. The change in enthalpy is manifested as a change in temperature of the stream, from T^i to T^f, and is governed by the following relationship:

$$\Delta H = mc_p(T^f - T^i) \tag{2-2}$$

Combining Equations (2-1) and (2-2), we obtain,

$$Q = mc_p(T^f - T^i) \qquad\qquad (2\text{-}3)$$

where T^i is the initial temperature before the supply (or removal) of heat, and T^f is the final temperature after it. For the purpose of this work, the product of the flow rate m and the specific heat c_p can be treated as a single entity, termed as heat capacity flow rate, denoted by mc_p. Note that both Q and $(T^f - T^i)$ in Equation 2-3 are positive when heat is supplied to a stream and negative when heat is removed, since the temperature will increase in the former case and decrease in the latter case. For the initial temperature equal to the source temperature and the final temperature equal to the target temperature, Equation 2-3 gives the total amount of heat required to be supplied to a cold stream (positive) or removed from a hot stream (negative). The absolute value of this amount of heat is called the *heat duty* of the stream.

When a hot stream is used to heat a cold stream, a certain amount of heat gets "transferred" from the former to the latter. Just as a liquid flows naturally from a higher level to a lower level, heat "flows" from a higher temperature to a lower one. Greater the temperature difference between the two streams, faster is the rate of heat transfer between them. This temperature difference is termed as the *driving force* for heat transfer (or heat exchange); it is denoted by ΔT. According to the theory of heat transfer, the rate of heat flow (per unit area of the heat exchange surface) and therefore, the amount of heat transferred, is directly proportional to this driving force.

8

Thus, when the driving force approaches zero, in the limiting case, the rate of heat transfer also approaches zero. From this standpoint, it is desirable to have as high a driving force as possible. On the other hand, the second law of thermodynamics suggests that for the maximum utilization of the heating and cooling potentials, and therefore, for the least amount of external heating and cooling requirements, the driving force should be as small as possible. Thus, we have two conflicting effects: higher driving force, which leads to smaller heat exchangers, reduces the capital cost of the network, whereas lower driving force reduces the operating cost of the network by reducing the amount of external heating and/or cooling required for the network. Since the operating cost is substantially higher than the capital costs, smallest permissible driving force is usually preferred by the designers. In practice, to prevent excessively large heat exchangers, a certain threshold value is specified as the minimum acceptable driving force. No two streams which have a driving force lower than the minimum value, ΔT_{min}, can be "matched" for heat transfer.

In dealing with the real world problems, usually one class of streams (hot or cold) have less total heat to be transferred than the other class. Additionally, there may be situations when a particular stream can not exchange heat with any other stream due to the minimum driving force constraint. To deal with such situations, "special" streams called *utility streams* (or simply, *utilities*) are employed for heat transfer. Two types of utilities, hot and cold, are available. A typical example of hot utility is steam and that of a cold utility is

cooling water. To differentiate with the utilities, the "original" streams are called *process streams*. Since the total cost of utilities constitute the major portion of the operating expenses for a heat exchanger network, it is desirable to minimize the utility consumption. In fact, the minimum utility consumption is a prime optimality criterion.

It is a usual design practice to assume that the utilities have extreme temperatures; i.e., steam has a temperature higher than any of the target temperatures of cold streams, and cooling water has a temperature lower than any of the target temperatures of hot streams. Therefore, any hot stream can be completely cooled by the cold utility (cooling water) and any cold stream can be completely heated by the hot utility (steam). To effectively utilize the heating and cooling potentials of the utilities to the maximum extent, their use is restricted as follows: a hot utility should be used to only to heat a cold process stream to its target temperature, and a cold utility should be used only to cool a hot process stream to its target temperature.

Another assumption made during the design of a heat exchanger network is that utilities do not change their temperatures during the heat exchange process. In reality there is a slight change, but it is negligible and does not affect the design or the performance of the final network. Consequently, the amount of utilities are measured in terms of their heat duties; their heat capacity flow rates (mc_p values) are not required for designing the networks. In contrast, process

streams change their temperatures during the heat exchange, the temperatures of hot streams decrease and those of cold streams increase.

Finally, to reflect the type of streams participating in any match, the resultant heat exchange units are classified into three categories. If both streams are process streams, then the unit is called a *heat exchanger*. If the hot stream is a utility (e.g., steam), it is classified as a *heater* and if the cold stream is a utility (e.g. cooling water), it is classified as a *cooler*. Obviously, it does not make any sense to match a hot utility with a cold utility for heat exchange. All three categories of units are generically referred to as *heat transfer units (HTUs)*. The amount of heat exchanged between the two streams in an HTU is called its *heat load*.

2.2 THE HEN PROBLEM

The heat exchanger network synthesis problem can be formulated as follows [see, e.g., Nishida et al., 1981; Mehta, 1986]:

Given a set of process streams, with specified flowrates and heat capacities, find an optimal set of heat transfer units (HTUs) that will transform the given source temperatures of all the streams to their respective desired target temperatures.

The following simplifying assumptions are usually made [Nishida et al. 1981; Jezowski and Hahne, 1986]:

(1) The utility streams, such as steam and cooling water are available at desired temperatures. The amounts are not

11

specified, but are assumed to be available in sufficient quantities.

(2) The utility streams do not change their temperatures during heat transfer.

(3) The heat capacities of all the streams are constant; they do not vary with the temperatures of the streams.

(4) Each HTU belongs to one of the three categories: a counter-current single-pass heat exchanger, a heater, or a cooler.

The following operational characteristics of a HEN are used for evaluating a candidate network.

(a) The annual investment and operational cost ($/year).

(b) The ease of instrumentation and control (*controllability*).

(c) The ability to survive through load fluctuations (*resiliency*).

(d) The ease of start-up and shut-down of the plant (*operability*).

(e) The modular network structure with interchangeable components (*flexibility*).

(f) Safety and reliability.

Except for the annual real cost ($/year), which is to be minimized, it is desired to maximize all other characteristics.

The synthesis or the preliminary design phase generates one or more candidate networks for detailed design phase. The total cost of a network is the only characteristic that can be estimated (within an accuracy of 20%) during the synthesis phase. All present automated synthesis systems attempt to generate the minimum cost structure. However, more often than not, this structure does not possess good (or acceptable) operational characteristics (b) through (f). Consequently,

12

alternate candidates, which have more than the minimum cost, are needed. It is for this reason that existing automated synthesis systems are not used by designers to solve complex industrial problems.

Experienced designers conceive several candidate structures, at least some of which have good operational characteristics (b) through (f) and, at the same time, none of them has excessively high cost [see, Barton et al., 1987]. These candidate structures are not generated by computing the estimated cost; instead, *they are based on the qualitative relationships between the structural and operational characteristics of HENs* [see, e.g., Nishida et al., 1981; Linnhoff et al., 1982]. Some of these *structural characteristics* are

(1) the number of HTUs in the network,

(2) the amount of utility consumption,

(3) the average driving force for each HTU in the network, and

(4) the distribution of heat loads of the HTUs in the network.

These relationships constitute the domain knowledge on which our AI approach is based to generate candidate structures in the synthesis phase of HEN design. At present, one such relationship is available from the literature [see, Nishida et al., 1981; Linnhoff et al, 1982; Linnhoff and Hindmarsh, 1983; Jezowski and Hahne, 1986]:

To generate a cost efficient (near minimum cost) heat exchanger network, the number of HTUs and the amount of utility requirement should be minimized.

For any given problem, several structures exist that satisfy this criterion. All of them have near minimum cost, only the other operational characteristics differ. Note that this set of structures

13

always contains the minimum cost structure, since it must also satisfy the same criterion. Therefore, this set of structures (or a subset of it) is usually preferred by the industrial designers as the starting point of for the next phase of design. Based on the foregoing analysis, the objective of the present work is to generate a set of candidate structures that satisfy the following optimality criteria:

(1) the minimum number of HTUs, and

(2) the minimum utility consumption.

It should be noted that the number of such structures, which is extremely small compared to the number of all possible structures, can be further reduced if and when additional relationships (domain knowledge) involving other structural and/or operational characteristics are available. Each additional piece of knowledge will make the task of HEN design less and less complex.

2.3 GRID DIAGRAM FOR NETWORK REPRESENTATION

The standard graphical representation scheme for a heat exchanger network (HEN) is the so-called *grid diagram*. Figures 2-1, 2-2 and 2-3 show typical grid diagrams with no network, partial network, and complete network, respectively. Each stream in a grid diagram is represented by a directed line from the source temperature to the target temperature, both of which are labeled at appropriate ends of the stream. All hot streams are drawn at the top with the source temperatures on the right-hand side and the target temperatures on the left-hand side, i.e. the hot streams "go" from the right to the left at

14

```
H1 (14.77) 148.9  ◄─────────────────────────── 271.1 (1804.9) H1

H2 ( 7.17)  82.2  ◄─────────────────────────── 198.9  (836.7) H2

H3 (10.53)  93.3  ◄─────────────────────────── 187.8  (995.1) H3


C1 ( 8.07)  37.8  ─────────────────────────►   226.7 (1524.4) C1

C2 (11.61)  82.2  ─────────────────────────►   226.7 (1677.6) C2

C3 (18.71)  60.0  ─────────────────────────►   160.0 (1871.0) C3
```
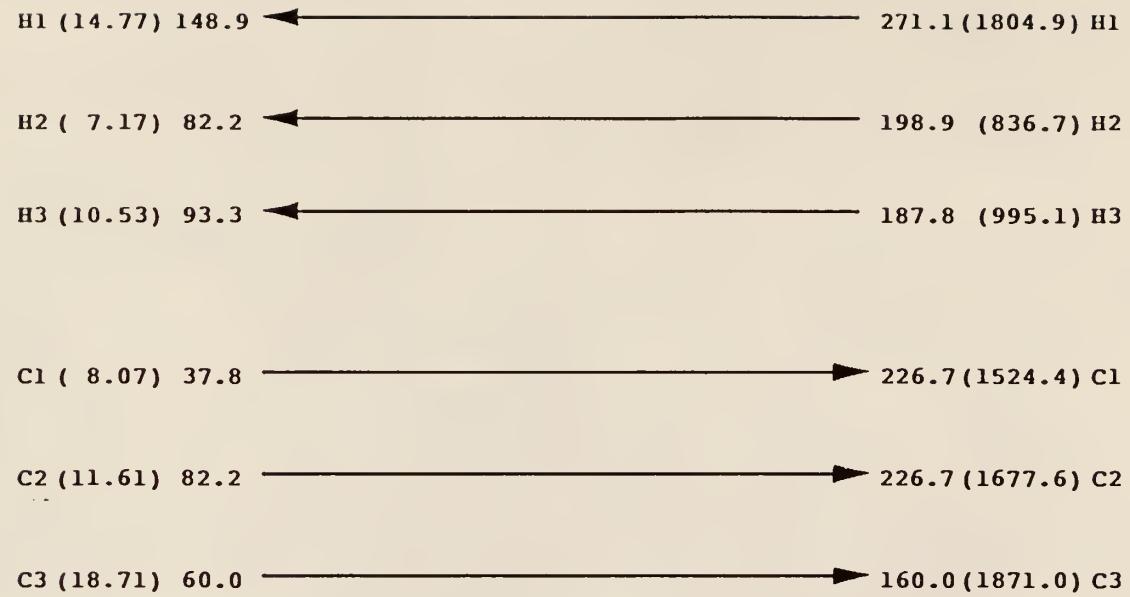
Fig. 2-1.   Grid diagram for a HEN problem at the
beginning of synthesis.
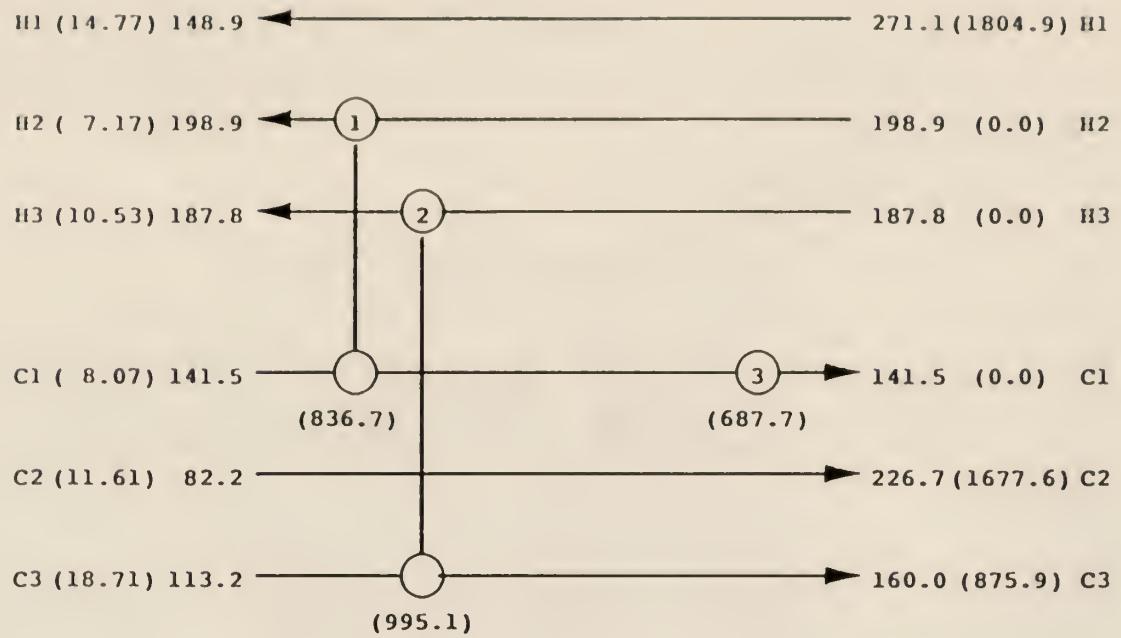
Fig. 2-2. Grid diagram for a partial solution of
the HEN problem in Fig. 2-1.
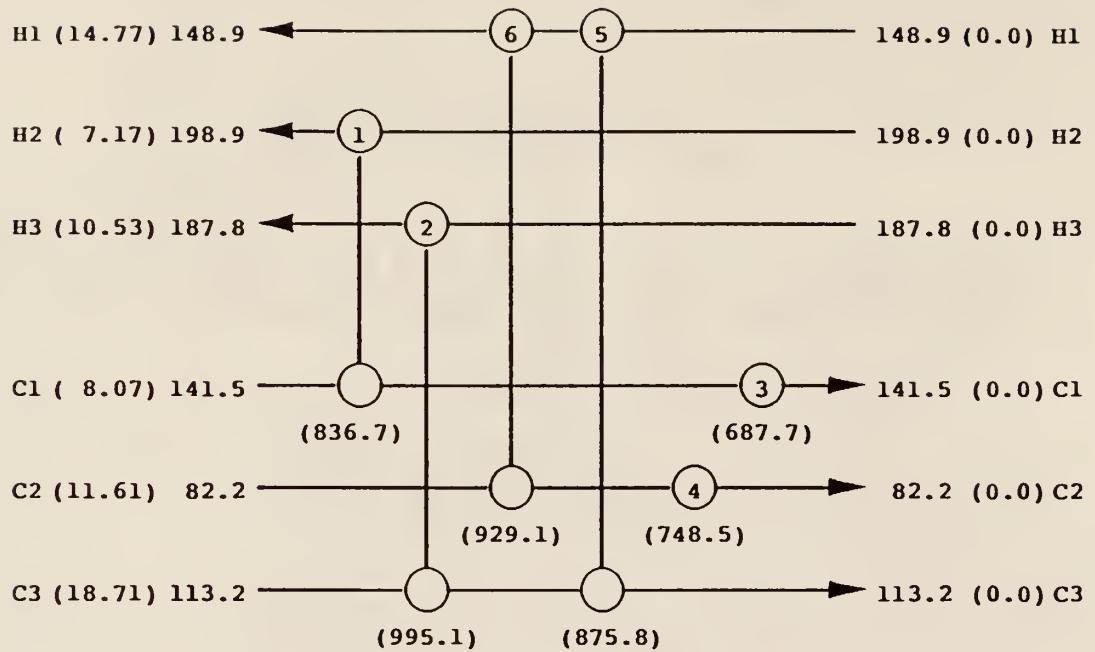
Fig. 2-3. Grid diagram for a complete solution of
the HEN problem in Fig. 2-1.

17

the top of the diagram. All cold streams are drawn at the bottom of the diagram with the source temperatures on the left-hand side and the target temperatures on the right-hand side, i.e., the cold streams "go" from the left to the right at the bottom of the diagram. Thus, in Figures 2-1, 2-2 and 2-3, there are three hot streams, H1, H2 and H3, and three cold streams, C1, C2 and C3. Note that for any stream, hot or cold, the higher temperature is at the right end of the diagram; therefore, it is called the hot end of the stream, and the corresponding temperature, the hot end (or hot side) temperature. Similarly, the lower temperature of any stream is on the left end of the diagram, which is called the cold end of the stream and the corresponding temperature, the cold end (or cold side) temperature. The temperatures of all streams increase from the left to the right, but not according to any scale. As the network is being synthesized, the amount of heat required to be supplied to or removed from a stream (i.e., its heat duty) changes. This causes changes in the hot and/or cold end temperatures of the streams. The temperatures at the two ends of a stream are the "current" values for the respective temperatures. This can be readily observed by comparing the Figures 2-1 and 2-2; the cold end temperatures of streams H2, H3 and C1 have changed from 82.2 to 198.8, 93.3 to 187.8 and 37.8 to 141.5, respectively. Also, the hot end temperature of C1 has changed from 226.7 in Figure 2-1 to 141.5 in Figure 2-2. In Figure 2-3, each stream has equal cold and hot end temperatures, indicating the end of synthesis.

Two additional values associated with each stream are displayed in the grid diagram; on the left edge of the stream is the heat capacity

18

flow rate, mc_p, (the product of flow rate and specific heat of the stream), and on the right is the *unsatisfied* heat duty of the stream. To distinguish these values from the source and target temperatures, they are parenthesized. Finally, the identification tag of a stream (e.g., H1, H2, ..., and C1, C2, ... etc.) is displayed at both the ends of the stream. Utility streams are not shown in the diagram. Usually, there is only one stream of each utility type, therefore, not showing them in the grid diagram is not likely to cause confusion. The identification tags are HU1 for the hot utility and CU1 for the cold utility. As shown in Figure 2-2 and 2-3, each HTU in the grid diagram is represented by a circle on the corresponding stream(s) with identification number (1, 2, 3, etc.) indicating the sequence in which the it has been created. A heat exchanger involving two streams is represented by a vertical line connecting the two circles on the corresponding streams, with the top circle containing the identification number. The heat load of a heater is displayed in parenthesis below the corresponding circle, that for a cooler is displayed above the corresponding circle, and for a heat exchanger, it is displayed below the "bottom" circle, i.e., the one on the cold stream. Each match (HTU) in a network is referred to by a name that is a concatenation of the hot and the cold stream (in that order) constituting the match, with a "/" in between. For example, a match between two streams H2 and C3 will be referred to as H2/C3, and a match between cold utility CU1 and hot stream H3 will be referred to as H3/CU1. The partial network shown in Figure 2-2 has two heat exchangers, corresponding to matches H2/C1 with

19

a heat load of 836.7 units, H3/C3 with a heat load of 995.1 units, and a
heater HU1/C1 with a heat load of 687.7 units.

Any HEN can be represented uniquely in a grid format. The units
for the values are not shown anywhere in the diagram; there are no
restrictions except that all the values must be in a consistent set of
units. Unless otherwise mentioned, the standard set of units will be
used throughout this work: Temperatures in °C, heat loads in kcal/hr
and heat capacity flow rates in kcal/hr-°C.

2.4 THE FRAMEWORK FOR SYNTHESIZING HEAT EXCHANGER NETWORKS

The solution process for HEN synthesis consists of two steps:
preanalysis and network invention. Preanalysis establishes the minimum
utility requirement for a given problem. This amount depends upon the
problem specifications (data) and is independent of the network
configuration. It is worth noting that this minimum utility requirement
is not just the net difference between the total heating needed for the
cold streams and the total cooling needed for the hot streams. The
target must also account for the minimum driving force constraint.
Well-established algorithms are available to predict this target [see,
e.g., Linnhoff and Flower, 1978; Cerda and Westerberg, 1980; Linnhoff
et al., 1982]. Therefore, in the present work, it is assumed that the
value of this target is known and available as part of the problem
specifications.

It is possible that some problems require both kinds of utilities,
hot as well as cold. In such cases, to ensure that the resultant

20

network does not violate the minimum utility requirement, the problem must be partitioned into two subproblems, each of which requires only one kind of utility (hot or cold) and must be solved independently by the network invention step. Final solution is obtained by putting the two subnetworks together. Once again, for the present work, it is assumed that such a partition, if required, has been already performed by the user. Thus, the scope of the present work is restricted to automating the task of network invention.

The network invention step is concerned with conceiving a network for a given problem (or a subproblem) with the minimum number of HTUs and featuring the minimum utility requirement as determined in the preanalysis step. It generates a network by sequentially matching pairs of streams; for each match it determines

(a) a pair of streams to be matched, and

(b) the extent and location of a match.

The extent of a match between a pair of streams is the amount of heat transferred in the match, i.e., the heat duty of the resultant HTU. The location of a match specifies the portions of the two heat duties that are matched. There are three possibilities for each heat duty, hot, intermediate and cold. Thus, there are nine possibilities for the location of a match for a selected pair of streams. The network invention procedure employed in the present work is based on the so-called elimination strategy [Linnhoff et al., 1982; Linnhoff and Hindmarsh, 1983; Mehta, 1986], which can be stated as follows:

To generate a network with the minimum number of HTUs and the

minimum utility consumption, each match between a pair of

21

stream must eliminate at least one of the streams and, if possible, both.

In essence, the elimination strategy specifies the restrictions on the extent and location of a match to ensure the optimality of the network being generated.

To ensure that the network being conceived features the minimum number of HTUs, the quantity of heat transferred in each HTU (i.e., the extent of each match) must be maximized. Obviously, the upper bound for this value is the smaller of the two heat duties of the streams being matched. Therefore, the maximum extent of heat transfer will reduce the heat duty of one of the stream to zero, thereby *eliminating* it from consideration for further matching. For the maximum extent of a match, the possible number of location of the match reduce to two. One location is hot end, where heat transfer begins at the hot ends of the two streams and terminates when one of the streams gets eliminated, thereby leaving the cold end of the other stream for further matching. The other match location is cold end, where the heat transfer begins at the cold end and once again, terminates when one of the streams get eliminated, thus leaving the hot end of the other stream for further matching. Figure 2-4 illustrates the hot and cold end matching for a pair of process streams. Note that a hot end match modifies (reduces) the hot side temperatures and a cold end match modifies (increases) the cold side temperatures of the process streams being matched.

As the heat is being transferred in an HTU, the driving force between the two streams (ΔT) may increase, decrease, or remain

22

H1 (14.77) 252.1 ◄—①——————————————————— 271.1 (280.5) H1

H2 (7.17) 82.2 ◄—————————————————————— 198.9 (836.7) H2

H3 (10.53) 93.3 ◄—————————————————————— 187.8 (995.1) H3

C1 (8.07) 226.7 ——————○——————————————► 226.7 (0.0)   C1
                  (1524.4)

C2 (11.61) 82.2 ——————————————————————► 226.7 (1677.6) C2

C3 (18.71) 60.0 ——————————————————————► 160.0 (1871.0) C3

(a) H1/C1 match at the hot end

H1 (14.77) 148.9 ◄——————————————①——————— 167.9 (280.5) H1

H2 (7.17) 82.2 ◄—————————————————————— 198.9 (836.7) H2

H3 (10.53) 93.3 ◄—————————————————————— 187.8 (995.1) H3

C1 (8.07) 37.8 ——————————————————○——► 37.8   (0.0)   C1
                               (1524.4)

C2 (11.61) 82.2 ——————————————————————► 226.7 (1677.6) C2

C3 (18.71) 60.0 ——————————————————————► 160.0 (1871.0) C3
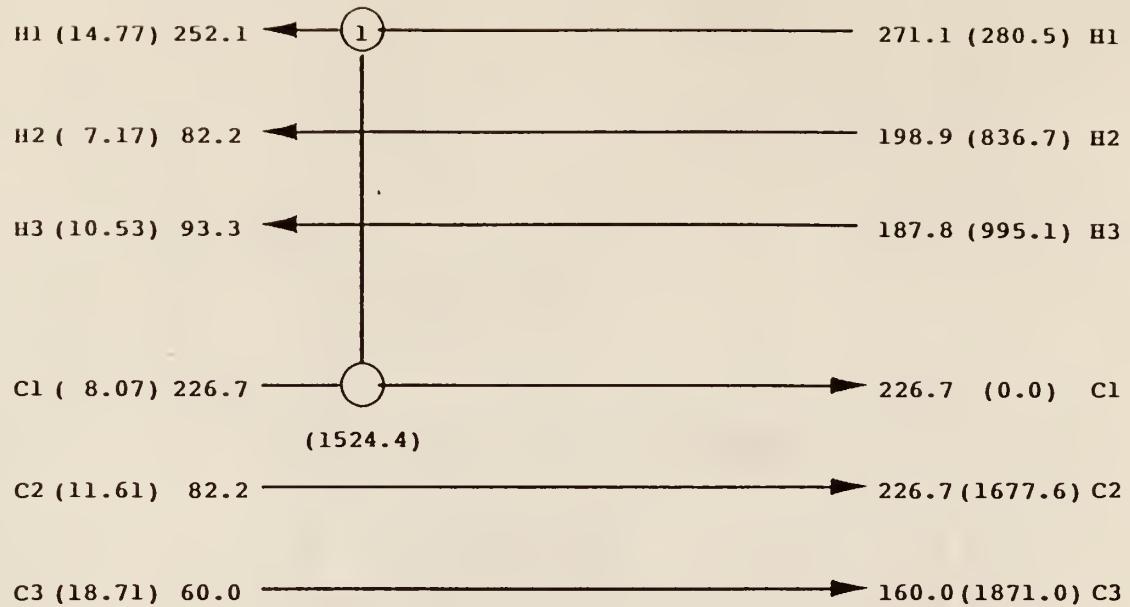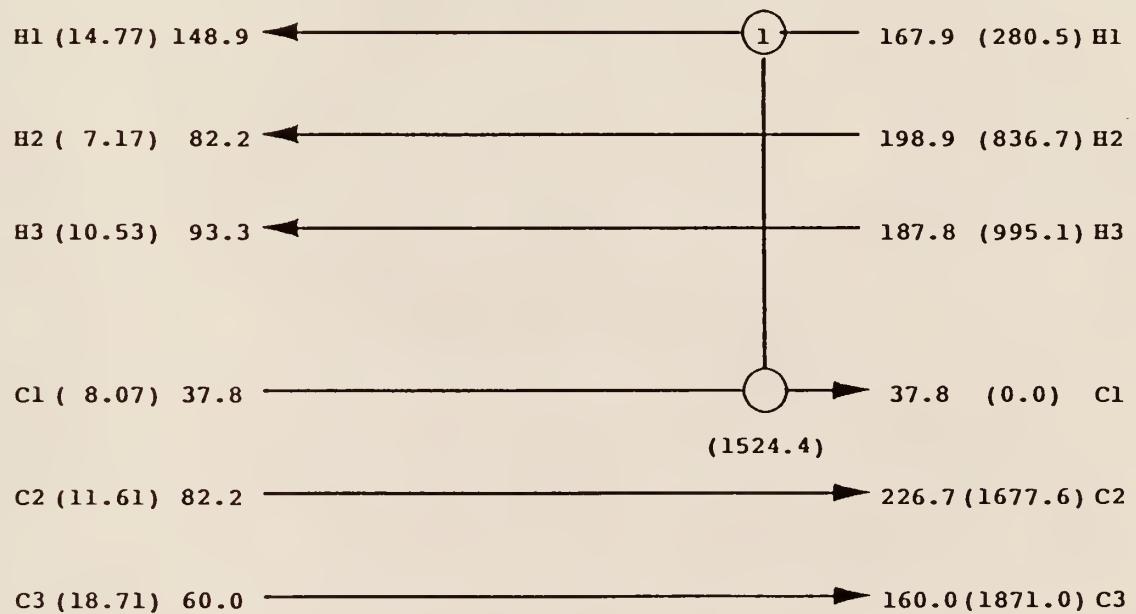
(b) H1/C1 match at the cold end

Fig. 2-4. Two ways of matching a pair of streams.

23

unchanged, depending upon the relative magnitudes of the mc_p values of the streams. Consequently, if both streams involved in a match are process streams, then there is a possibility that before the maximum possible amount of heat is transferred (i.e., before one of the stream gets eliminated), the minimum driving force constraint is violated. Note that such a situation does not arise when one of the streams is a utility stream, since is has been assumed that utility streams have "constant" temperatures that are sufficiently high (or low) to heat (or cool) any process stream. The necessary and sufficient conditions for ensuring that the maximum extent of heat transfer is possible without the violation of the minimum driving force constraint have been derived and are available in the literature [Mehta, 1986]. They are, for the hot end match,

$$\Delta T_{he} > \Delta T_{min} \qquad (2-4)$$

and

$$\left[\Delta T_{he} - min(Q_h, Q_c) \left\{ \frac{1}{(mc_p)_h} - \frac{1}{(mc_p)_c} \right\} \right] > \Delta T_{min} \qquad (2-5)$$

where

$$\Delta T_{he} = T_h^s - T_c^t \qquad (2-6)$$

and for a cold end match,

$$\Delta T_{ce} > \Delta T_{min} \qquad (2-7)$$

and

$$\left[\Delta T_{ce} - min(Q_n, Q_c) \left\{ \frac{1}{(mc_p)_h} - \frac{1}{(mc_p)_c} \right\} \right] > \Delta T_{min} \qquad (2-8)$$

where

$$\Delta T_{ce} = T_h^t - T_e^s \qquad\qquad (2-9)$$

In all the above relationships, Q, mc_P and T are the heat duty, heat capacity flow rate, and temperature of a stream, respectively; subscripts h and c designate the hot and cold streams, respectively; and superscripts s and t, the source and target values of temperatures, respectively. As long as a pair of streams selected for matching satisfy one of the above two sets of conditions, at least one of the streams will be eliminated and the maximum amount of heat will be transferred (equal to the minimum of the two values Q_h and Q_c).

It is possible to arrive at a situation during HEN synthesis where no two streams satisfy even one of the two sets of elimination conditions. In such situations, stream splitting is resorted to, rather than settling for a less-than-the-maximum value for the extent of a match. By splitting a stream, two or more substreams are generated; these substreams have lower values of heat capacity flow rates (mc_P values), thereby enhancing the possibility of satisfying the elimination conditions. Note that the sum of the mc_P values for all the substreams add up to the mc_P value of the "parent" stream, and therefore, the substreams can be merged any time to yield the original stream (albeit with the modified temperature). To obtain the minimum number of units in the resultant HEN, each substream must eliminate all the streams with which it is matched. The task of stream splitting has not yet been formalized, therefore sufficient amount of domain knowledge is not available to carry out this task. Most existing methods do not

25

split streams without the user's help. The user specifies the manner in which the splitting is to be accomplished. The details of how to carry out this task are beyond the scope of this work and hence will not be pursued here. For the purpose of this work, the information regarding which stream to split and what substreams to be generated will be assumed to be available, whenever required.

CHAPTER 3. AN AI APPROACH TO THE HEN SYNTHESIS PROBLEM

Several methods have been employed for computer-based synthesis of
heat exchanger networks. These methods are based either on numerical
techniques such as linear programming [Kesler and Parker, 1969;
Kobayashi et al., 1971; Cena et al., 1977] and mixed integer linear
programming [Papoulias and Grossmann, 1983], or on search techniques,
such as total enumeration [Pho and Lapidus, 1973], branch and bound
[Rathore and Powers, 1975; Greenkorn et al., 1978; Grossmann and
Sargent, 1978] and depth-first branch and bound [Jezowski and Hahne,
1986].   Each of these methods yield only one candidate structure,
namely, the one having the minimum estimated cost ($/year).   Finding
such a specific solution is a task of considerable complexity, since
enormous number of feasible network structures exist for a given
problem.   Since the estimated cost is only 20% accurate and the minimum
cost network may not possess acceptable operational characteristics,
there is a need to generate alternate structures which have near minimum
cost.   These networks can then be analyzed and evaluated in the detailed
design phase.

Numerical techniques based methods may be employed for generating
successive minimum cost networks, provided each time the problem is
reformulated with added constraints prohibiting the previously obtained
configurations.   Such an approach, though potentially feasible, is not
pragmatic; finding a single solution is too cumbersome to repeat the
method multiple times. For example, Papoulias and Grossmann [1983] have
reported an MILP formulation that requires 30 binary variables, 172

27

continuous variables and 119 constraints for a ten stream problem (the so-called 10SP1 test problem). The methods based on search techniques can be modified to generate multiple candidate networks. However, they do not employ domain knowledge to focus their search, thereby, end up searching the entire search space. For example, Jezowski and Hahne [1986] have reported generating over 10,000 nodes for the 10SP1 test problem, and close to 100,000 nodes for a problem with twenty streams. As evident from this analysis, the complexity of the conventional methods increase sharply when multiple candidate structures are required. To overcome this complexity, we propose an AI based approach, which utilizes the available domain knowledge. We employ heuristic search technique based on the elements of domain knowledge presented in sections 2.2 and 2.4 to reduce the search for the desired network configurations.

3.1 SEARCH SYSTEMS

A search system associated with a problem has three components [see, e.g., Nilsson, 1980; Barr and Feigenbaum, 1981; Rich, 1983]:

(1) a *database*, which describes both the current task-domain situation and the goal (the solution);

(2) a set of *operators* to manipulate the database; and

(3) a *control strategy* for deciding what operator to apply and where to apply it.

The successive applications of operators to the current task-domain situation to produce a modified situation, is called a *forward reasoning*

strategy. Thus, a forward reasoning strategy starts with the initial problem configuration and transforms it into a goal configuration. On the other hand, a *backward reasoning* control strategy applies operators to the goal (i.e., the solution configuration) to produce one or more subgoals whose solutions will lead to the solution of the original problem. Each of these subgoals then becomes a current goal and is treated in similar fashion. Thus, a backward reasoning strategy starts with the solution configuration and by recursive applications of the available operators, arrives at the initial problem configuration. For complex problems, the two strategies can be combined to form a *bidirectional* or *opportunistic reasoning* strategy. Often, the forward reasoning is called *data directed* or *bottom-up* strategy, whereas the backward reasoning is called *goal directed* or *top-down* strategy. Obviously, the operators needed for the two strategies are of different types.

Any goal-oriented problem can be solved using a search system; the problem is formulated as a *state-space search* problem. The search space is perceived as consisting of a set of all possible problem states, including the start state(s), goal state(s) and all the intermediate states, and a set of operators for state transformations. Each operator acts upon one state (the "current" state) to produce one or more "new" states. Forward, backward, or bidirectional reasoning strategy may be employed to "move" around in the state-space of the problem.

Finding a solution in a search system can be modeled as the traversal of a directed graph in which each node represents a state and each arc represents the operator that transforms one operator into

29

another state. The search system must find a path through the graph, starting at the initial (start) state and ending in one or more final (goal) states. Since the graph to be searched can, in principle, be generated from the operators, the graph is said to be implicitly represented by the operators. Only those parts of the graph that need to be searched are generated explicitly (i.e., actually constructed by the system). The size of the graph actually constructed by a system depends upon the nature and extent of the search, which, in turn, is determined by the control strategy. For this reason, the control strategy is often referred to as the *search technique* or the *search strategy*.

General-purpose search techniques include generate-and-test, hill climbing, breadth-first, best-first, problem reduction, constraint satisfaction and means-ends analysis. (For a detailed discussion of these techniques, see Barr and Feigenbaum [1981] or Rich [1983]). All these techniques are more or less independent of any particular task or problem domain. The past decade of AI research has revealed the inability of these general-purpose methods to efficiently solve complex real world problems. More often than not, these techniques suffer from combinatorial explosion of the search space (search space grows exponentially with the size of the problem). Therefore, they are called *weak methods* in AI literature.

By early 1970's, the AI researchers realized that a good search strategy should use some form of knowledge about the problem domain for efficiently solving the problem. The performance of a weak method can be significantly improved by using the domain knowledge to appropriately

guide the search. Thus, weak methods provide a framework into which domain knowledge can be placed to create powerful problem-specific strategies to solve complex problems. This shift in approach has given considerable impetus to the research on knowledge representation, resulting in several formalisms, such as predicate calculus, production systems, frames, scripts, conceptual dependencies and conceptual graphs. The selection of any one formalism for solving a problem depends on the nature and characteristics of the problem and the domain knowledge. To exploit the fullest power of these formalisms, various programming paradigms and environments have been suggested, including the logical programming, functional programming and object-oriented programming paradigms.

3.2 A SEARCH SYSTEM FOR THE HEN SYNTHESIS PROBLEM

The search system formulated in the present work differs from the systems discussed at the beginning of this chapter in the following ways.

(a) It is capable of providing several candidate network structures, all featuring the minimum number of HTUs and the minimum utility consumption without searching the space multiple times.

(b) It uses the domain knowledge, in the form of the elimination strategy and the associated necessary and sufficient conditions (cf. section 2.4), to focus the search in the restricted region of the search space.

31

(c) It does not use the estimated cost of the network ($/year) as the basis for evaluating the potential candidate structures.

(d) It permits stream splitting to generate spilt structures with the minimum number of units and minimum utility consumption, when unsplit solutions are not feasible. In such cases, the split structures generated by the present system will have lower cost than the unsplit structures reported by the other methods.

(e) It employs a heuristic control strategy that utilizes the domain knowledge in an attempt to prevent the generation of infeasible structures, thus focusing the search and minimizing the backtracking.

In the succeeding subsections, we formulate the HEN synthesis task as a state-space search problem and define the three components that constitute the search system. Furthermore, we show how domain knowledge can be incorporated into this framework to restrict the search space and to guide the control strategy.

3.2.1 State-Space Representation of HEN problem

A "state" of a HEN synthesis problem consists of the characteristic values (the source and target temperatures, the heat duty, and the specific heat flow rate) of all the streams in the problem, as well as the feasibilities of matching all the pairs of "uneliminated" streams. The states are transformed by four operators; an operator applied to a state modifies both, the characteristic values of one or more streams and the feasibilities of matching this (or these) stream(s). The

states, which constitute the search space, can be classified into the following four categories:

(a) the initial problem state or the start state, where no matches have been made and consequently, the heat duty of none of the streams is satisfied, either partially or completely (i.e., the heat duty of every stream is the same as the initial value),

(b) the solution states, where the heat duty of every stream is completely satisfied (i.e., the heat duty of every stream is zero),

(c) the intermediate states where heat duties of some (but not all) of the streams are satisfied, either partially or fully, and

(d) the dead-end states, where heat duties of one or more streams can not be satisfied completely with any match.

All these states constitute an implicit form of a directed graph. Only those portions of this graph, which are explored by the control strategy, are explicitly generated by applying the operators described in section 3.2.2. Each arc connecting two states in the explicit form of the search graph can be perceived as representing an operator that transforms the source state into the destination state.

3.2.2 Operators

The HEN synthesis problem requires only four operators. These operators and the effect they produce are as follows:

MATCH: Makes a match between the specified pair of streams at the specified end as selected by the control strategy. The extent of a match is the maximum possible value, i. e., the lower of the heat duties of the two streams being matched, in accordance with the elimination strategy. As a consequence of the match, an HTU is "produced" for the match between these two selected streams, and the heat duties of the two streams get reduced by an amount equal to the HTU load. Also, the process streams that take part in the match (at least one, at most two) change their characteristic values; for a hot end match, the hot side temperature(s) get reduced, whereas for a cold end match, the cold side temperature(s) get increased. In the event that the extent of match equals the residual heat duty of any of the two constituent streams (before the match), this stream gets eliminated and if it is a process stream, then the hot and cold side temperatures will become equal.

For example, each of the partial networks in Figure 2-4 are obtained by a single application of MATCH operator to the start state of the problem in Figure 2-1. In each case, the operator acts on the same pair of streams (H1 and C1), but at different ends. In both cases, the match generates a heat exchanger with a load of 152.4 units, eliminates C1 (heat duty becomes 0.0) and reduces the heat duty of H1 from 1804.9 in Figure 2-1 by an amount equal to 1524.4 units to 280.5 in Figure 2-4. However, the temperature changes depend on the end at which the operator is applied. In case (a), the operator MATCH is applied at the cold end, resulting in the change of cold end (left edge) temperatures of H1 (from 148.9 in Figure 2-1 to 252.1 in Figure 2-4) and C1 (from 37.8 in Figure 2-1 to 226.7 in Figure 2-4). In case (b), the operator MATCH is applied

34

at the hot end, resulting in change of hot end (right edge) temperatures of H1 (from 271.1 to 167.9) and C1 (from 226.7 to 37.8). Note that since C1 has been eliminated, its hot and cold end temperatures are identical in each case, 226.7 in (a) and 37.8 in (b). The difference in the values between (a) and (b) is due to the difference in the manner of application of the MATCH operator.

UNMATCH: Produces exactly the opposite effect of the MATCH operator. This operator is essential for back-tracking the solution steps, should a dead-end state be encountered. The restriction for applying this operator is that only those streams which were matched by the latest application of the match operator are eligible. In other words, at any instant, only the last match can be "undone." Nevertheless, by successively applying this operator, several matches can be undone, in the reverse order of their making. Note that successive application of MATCH and UNMATCH operators will bring the problem back to the same state.

SPLIT: Splits the specified stream into the specified number of substreams. The value of the heat capacity flow rate for each stream is specified by the control strategy as a fraction of the corresponding value for the original or parent stream. Note that this operator can act upon either a process stream, or a utility stream. Splitting a stream temporarily disables that stream for further operations; only the substreams can be operated upon. Each substream inherits both the hot and cold side temperatures from its parent stream. Finally, at all

35

times, each substream retains the identity of its parent stream, and the parent stream retains the identify of all its substream.

MERGE: Merges the specified substreams at the specified end (hot or cold) to form either the parent stream (when all the substreams are merged) or a *composite substream* (when only some of the substreams are merged). The substreams that are merged are disabled for any subsequent operations. If merging results in the parent stream, then it is reactivated for subsequent operations. The temperature of the resultant stream is obtained by taking the weighted average of the corresponding substream temperatures, the weight factor being the corresponding heat capacity flow rates. If the MERGE operator is applied immediately following an application of the SPLIT operator, then merging all the substreams returns the problem back to the same state (the source state for the SPLIT operator). In contrast, if one or more applications of MATCH operator separates the application of SPLIT and MERGE operators, then the destination state will not be the same as the source state for the SPLIT operator when all the substreams are merged.

3.2.3 Use of Domain Knowledge to Restrict the Search

Even for an average size problem, the search space described in the preceeding subsection is too large and complex to handle. For example, consider a problem with 10 streams. For simplicity, the following discussion is restricted to only one operator, MATCH. Furthermore, suppose that the extent of match is the maximum amount of heat transfer

36

permissible between a pair of streams. In a "blind" search strategy, each of the 10 streams can be matched with 9 others (a stream can not be matched with itself!) at either of the two ends, hot and cold. Thus, for the start state, we have 180 immediate successors. Of course, as we move towards one of the goal states, this number reduces all the way to 1. Nevertheless, the search graph generated for the problem under the restrictions stated in 3.2.2 is too large to be efficiently handled. Rather than burdening the search technique (control strategy) with the task of selecting the successor states from this enormous number of possibilities, the available domain knowledge can be utilized to reduce the possible number of successor states to a minimum. The elimination strategy can restrict the search in the following four ways.

(A) From the discussion in section 2.1, we know that the intent of matching a pair of streams is to use the hot stream to heat the cold stream. To utilize this knowledge, we can divide the set of streams (process streams as well as utility streams) into two subsets: hot streams and cold streams. Now, for matching, only one stream from each category needs to be considered; furthermore, hot utilities must not be matched with the cold utilities. These constraints substantially reduce the possible number of immediate successor states that need to be considered for traversing the graph. Suppose that the 10 stream problem considered earlier in this subsection consists of 6 hot streams and 4 cold streams, with one utility each. Now the number of immediate successor states that need to be considered are 46 (23 pairs of streams with possible matches at both ends). Compared

37

to the previous figure of 180, we can see that the search space has been reduced to one fourth!

(B) The second element of the domain knowledge restricts the the number of successor states to those which do not violate the optimality criteria (the minimum number of HTUs and the minimum utility consumption). To ensure this, it is necessary to follow the elimination strategy, i.e., each match must eliminate at least one of the streams, and if possible, both (cf. section 2.4). The necessary and sufficient conditions for following this strategy (Equations 2-4 and 2-5 for a hot end match and Equations 2-7 and 2-8 for a cold end match) depend only on the characteristic values of the two streams being matched. Consequently, we can put a restriction that a match between a pair of streams is *feasible* at hot and/or cold end if and only if the corresponding elimination conditions are satisfied. Obviously, this will maximize the extent of the match. Now only those pairs of streams, for which the match is feasible at least at one of the ends, need be considered. Note that whenever any characteristic value of any one of the stream changes, the feasibilities of all the matches involving this stream, must be re-evaluated. On the other hand, if none of the characteristic values of any of the two streams of a match change, then its feasibility remains unaltered.

(C) The third element of the domain knowledge exploits the nature of the utility streams. Based on the assumptions (a) and (b) in section 2.2, we can conclude that a hot utility can, at all

38

times, be used to heat any cold process stream to its target temperature, and a cold utility can, at all times, be used to cool any hot process stream to its target temperature. Furthermore, as stated in section 2.4, for utilizing the maximum heating and cooling potentials, and thereby not violating the minimum utility consumption constraint, a hot utility must be matched with a cold process stream only at its hot (target) end, and a cold utility must be matched with a hot process stream only at its cold (target) end. This gives us the following "rule" for determining the feasibility between a utility stream and a process stream. Any match between a hot process stream and a cold utility stream is always feasible at the cold end and never feasible at the hot end, whereas any match between a cold process stream and a hot utility stream is always feasible at the hot end and never feasible at the cold end. Once again, the extent of a match equals the minimum of the heat duties of the two streams being matched, in accordance with the elimination strategy.

(D) The fourth and the final element of the domain knowledge is that an eliminated stream should not be considered at all for any further application of any operator.

Out of the above four elements of the domain knowledge, (A) and (D) arise out of the basic background of the domain, whereas (B) and (C) arise solely out of the elimination strategy chosen for the present work. The extent of reduction of the search space due to this additional knowledge (elements B, C and D) can not be predicted apriori; it depends on the characteristic values of streams constituting the

39

problem. Usually, the reduction is greater for larger problems. In any event, this reduction is always significant, ranging from 20 to 80 percents.

The grid diagram described in section 2.3 is not equipped to portray the restricted nature of the search space. An alternate, better representation, called *match matrix*, has been chosen in the present work for displaying the state of a HEN problem. This representation is a significantly modified version of the original one proposed by Pho and Lapidus [1973]. The modifications enable a match matrix to display the domain knowledge required by the control strategy in determining the subsequent applications of operators to generate immediate successors in the restricted search space.

Each of the Figures 3-1, 3-2 and 3-3 is an example of a grid diagram and the corresponding match matrix for the start state, an intermediate state and a solution state for a HEN problem with six streams. Note that the grid diagrams in these figures are identical to the ones in Figures 2-1, 2-2 and 2-3, respectively. Each row of a match matrix contains the match information for a cold stream, and each column, the match information for a hot stream; the rows and columns are labeled with the corresponding stream "names". Each entry (cell) in the matrix, belongs simultaneously to a row and a column. Therefore, a cell displays the information pertaining to the match between the cold and hot streams corresponding, respectively, to the row and column to which it belongs. If a match already exists between two streams, then the corresponding cell contains the heat load of the resultant HTU; otherwise, it contains the feasibilities of matching the corresponding

40

H1 (14.77) 148.9 ◄──────────────────────── 271.1 (1804.9) H1

H2 (7.17) 82.2 ◄──────────────────────── 198.9 (836.7) H2

H3 (10.53) 93.3 ◄──────────────────────── 187.8 (995.1) H3

C1 (8.07) 37.8 ──────────────────────────► 226.7 (1524.4) C1

C2 (11.61) 82.2 ──────────────────────────► 226.7 (1677.6) C2

C3 (18.71) 60.0 ──────────────────────────► 160.0 (1871.0) C3
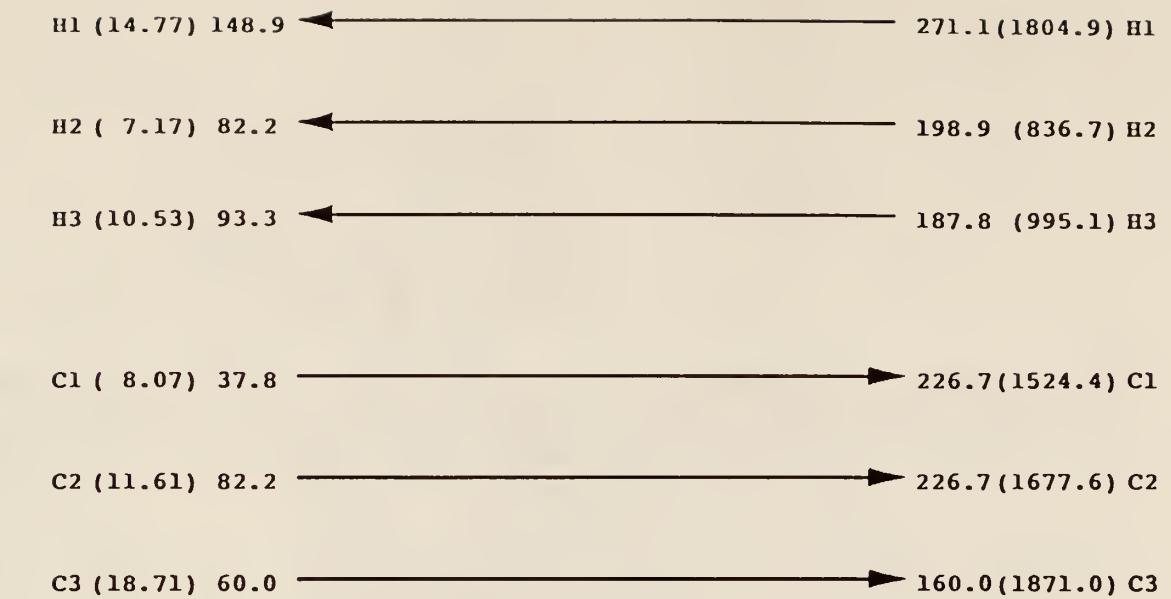
(a) Grid Diagram

Hot Cold	H1	H2	H3	HU1	Qc
C1	H C	* C	* C	H *	1524.4
C2	H C	* *	* *	H *	1677.6
C3	H C	* C	* C	H *	1871.0
Qh	1804.9	836.7	995.1	1436.2	

(b) Match Matrix

Fig. 3-1. Start state for a HEN synthesis problem.

H1 (14.77) 148.9 ◄────────────────────────────── 271.1 (1804.9) H1

H2 (7.17) 198.9 ◄──①──────────────────── 198.9 (0.0) H2

H3 (10.53) 187.8 ◄──────②────────────── 187.8 (0.0) H3

C1 (8.07) 141.5 ──────◯──────────③──► 141.5 (0.0) C1
          (836.7)   (687.7)

C2 (11.61) 82.2 ──────────────────────► 226.7 (1677.6) C2

C3 (18.71) 113.2 ──────◯──────────────► 160.0 (875.9) C3
        (995.1)
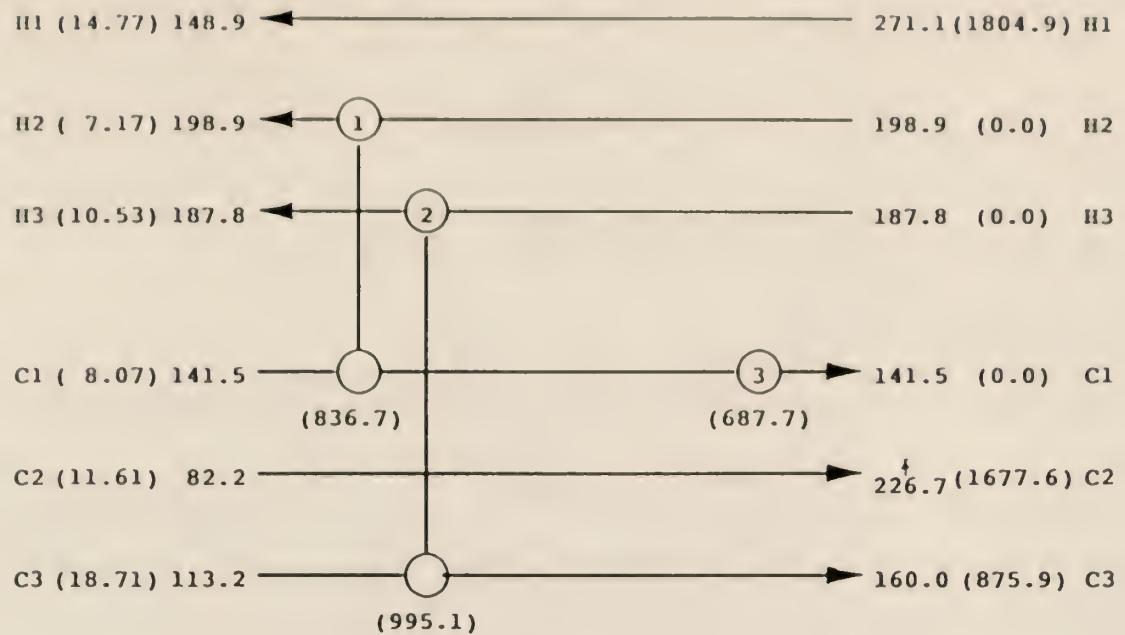
(a) Grid Diagram

Hot / Cold	H1	H2	H3	HU1	Qc
C1	----	836.7	----	687.7	0.0
C2	H C	----	----	H *	1677.6
C3	H C	----	995.1	H *	875.9
Qh	1804.9	0.0	0.0	748.5	

(b) Match Matrix

Fig. 3-2. Intermediate state for the HEN synthesis problem in Fig. 3-1.

H1 (14.77) 148.9 ◄─────── ⑥ ─ ⑤ ─────────── 148.9 (0.0) H1

H2 (7.17) 198.9 ◄── ① ─────────────────── 198.9 (0.0) H2

H3 (10.53) 187.8 ◄───── ② ─────────────── 187.8 (0.0) H3

C1 (8.07) 141.5 ───────○───────────③─► 141.5 (0.0) C1
(836.7)   (687.7)

C2 (11.61) 82.2 ─────────────○──────④─► 82.2 (0.0) C2
(929.1)   (748.5)

C3 (18.71) 113.2 ───────○───────○──────► 113.2 (0.0) C3
(995.1)   (875.8)
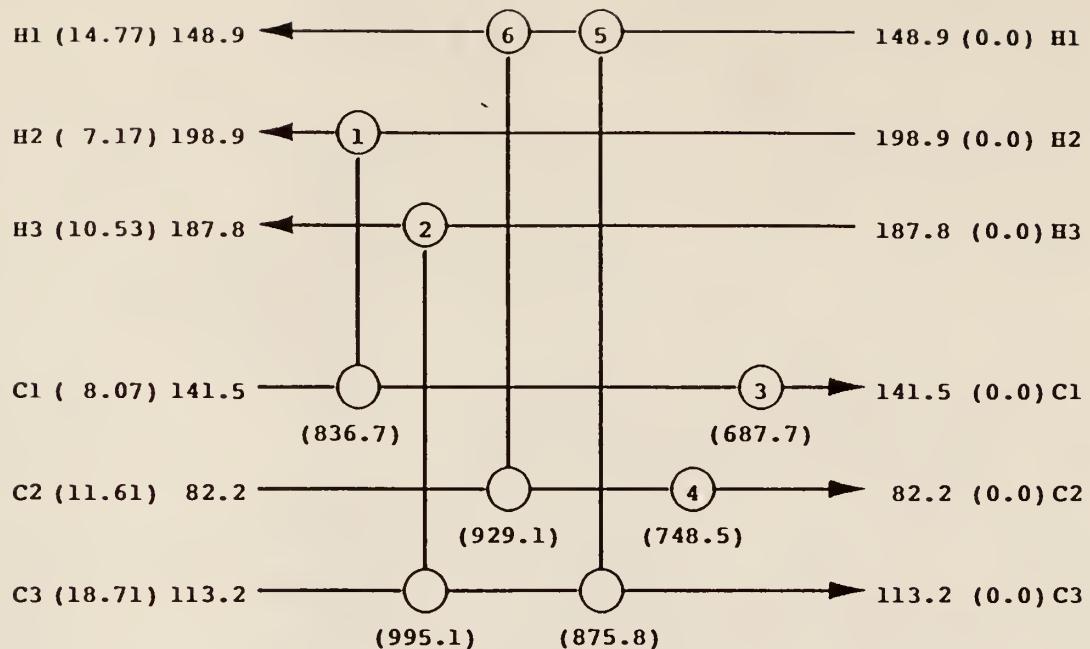
(a) Grid Diagram

Hot / Cold	H1	H2	H3	HU1	Qc
C1	----	836.7	----	687.7	0.0
C2	929.1	----	----	748.5	0.0
C3	875.8	----	995.1	----	0.0
Qh	0.0	0.0	0.0	0.0	

(b) Match Matrix

Fig. 3-3. Solution state for the HEN synthesis
          problem in Fig. 3-1.

pair of streams. If the heat duty of one of the stream constituting a match is zero (i.e., if it has been eliminated), then the feasibility of the match is no longer required. Such a cell contains a dash ("——"). If neither stream (constituting the match) has been eliminated, then the cell contains the feasibility as determined by the domain knowledge (B) and (C) described in the preceding paragraphs. The first symbol in the cell corresponds to the hot end match; the feasibility is indicated by an "H" and the infeasibility, by a "*". The second symbol corresponds to the cold end match; the feasibility is indicated by a "C" and the infeasibility, by a "*". The last row, labeled Qh, contains the unsatisfied or residual heat duties of the hot streams (i.e., the heat duties yet to be satisfied). Similarly, the last column, labeled Qc, contains the unsatisfied or residual heat duties for the cold streams.

The match matrix in Figure 3-1(b) shows that no match has yet been made. Thus, it is a start state. The match matrix in Figure 3-2(b), which corresponds to an intermediate state because some, but not all, of the streams have been eliminated. It presents the following information pertaining to the state of the problem.

(a) The partial network consists of three HTUs; one heat exchanger each for H2/C1 and H3/C3 matches, with heat loads of 836.7 and 995.1 units, respectively, and a heater for HU1/C1 match, with a heat load of 687.7 units.

(b) Three streams have been eliminated from the problem: hot streams H2 and H3, as well as cold stream C1.

(c) The matches H1/C2 and H1/C3 are feasible at both, hot and cold, ends whereas HU1/C2 and HU1/C3 matches are feasible only at the hot ends.

From (a) and (b), it can be confirmed that the partial solution has been able to adhere to the elimination strategy. In other words, up to this point in synthesis, the solution has managed to have only the minimum number of HTUs, independent of the path traversed in arriving at this state. The information in (c) helps the control strategy in deciding what operators should be applied and to which streams should they be applied; e.g., which streams should be matched next, H1/C2, H1/C3, HU1/C2 or HU1/C3. It is clear from this example that a match matrix displays the current state of a problem in a form which facilitates the decision-making by the control strategy. The match matrix in Figure 3-3(b) has no feasible matches. Since the heat duties of all the streams (in the last row and last column) is zero, this is a solution state.

3.2.4 Control Strategy

In this subsection, we describe the control strategy employed in the present work, i.e., we specify which of the four operators is to be applied to a problem state and which streams does the operator act upon. The domain knowledge described in section 3.2.3 is used to restrict the generation of explicit search graph by preventing the control strategy from obtaining a structure that does not satisfy the two optimality criteria (the minimum number of HTUs and the minimum utility

45

consumption). However, this search space contains an enormous number of dead-end states, in addition to all the desired solution states. Therefore, for a control strategy to be efficient, it must prevent as many dead-end states as possible. The present control strategy attempts to prevent dead-ends by examining the feasibilities of all matches. This procedure selects one or more "most constrained" pairs of streams which, if not matched, are likely to lead to a dead-end. At each step of the search, these most constrained pairs of streams form the alternate selections for applying the MATCH operator. These alternate selections will be explored in the depth-first fashion to generate all possible candidate networks. The detailed descriptions of the two components of the control strategy are as follows.

Selection of Operator: MATCH is the default operator. As long as this operator is applicable, no other operator will be considered. In other words, this operator will be applied till a solution state or a dead-end state is reached. For each application of MATCH operator, one of the most-constrained pairs of stream will be chosen at random; the others will be "remembered" for future exploration of alternate paths. For this purpose, a pair of streams which can be matched at both the ends, hot and cold, are considered as separate choices. The extent of the match, i.e., the heat load of the resultant HTU will be in accordance with the elimination strategy, as specified in section 3.2.2.

In the event that there are no more feasible matches in the match matrix, a dead-end or a solution state is reached. After informing the user about the situation, the UNMATCH operator will be applied. Since

46

only the last match can be undone, the question of selection of streams does not arise. The match undone will be remembered and will not be considered ever again. After one application of UNMATCH, the MATCH operator will be considered again, this time the selection will be from the remaining pairs of most constrained streams for that state. Application of UNMATCH operator is essentially a backtracking step in traversing the search graph. If there are no more pairs of streams available as alternate matches after an application of UNMATCH operator, then the UNMATCH operator will be applied recursively until an alternate pair of streams can be found. If no amount of backtracking (unmatching) yields an alternate pair of streams for matching, and not a single solution state has been reached, then the operator SPLIT will be applied. The choice of a stream to be split and the manner of splitting will be obtained from the user, as the adequate knowledge is not available to define a formal splitting strategy. Once again, this knowledge can be eventually incorporated when it is available. Once a stream is split, all the resultant substreams should be eliminated by applying the match operator. When all substreams have been eliminated, they are recombined together using the MERGE operator. At any given time, only one stream can remain split; only when it is merged back, can another stream be split.

Selection of Streams: Amongst all pairs of streams that constitute feasible matches (for a particular state), those which are the most constrained, are selected for making the next match. The following stepwise procedure is employed for the selection.

(1) Determine the number of feasible matches for each stream. For this purpose, a hot end match and a cold end match between the same two streams are considered as two matches. Essentially, this amounts to counting the numbers of H's and C's in each row (for cold streams) and each column (for hot streams). Thus, associated with each stream is the number of feasible matches into which it can participate.

(2) Select the stream(s) having the lowest non-zero number of feasible matches that it (they) can participate into.

(3) From all the streams that can be matched with the stream selected in (2), select the one that has the least non-zero number of feasible matches.

(4) If only one stream is selected in (2) and only one in (3), then these two streams constitute the most constrained pair of streams. If more than one stream get selected in (3), but not in (2), then each stream selected in (3), together with the one selected in (2), constitutes a most constrained pair of streams.

(5) If more than one streams are selected in (2), then for each of them, obtain the most constrained pairs of streams using (3) and (4) above. From these pairs, those that have the least total number of feasible matches constitute the set of most constrained pairs of streams.

The list of most constrained pairs of streams obtained as above is used by the control strategy to restrict the application of MATCH operator to focus the search.

CHAPTER 4.   HENSYN: AN IMPLEMENTATION USING LOOPS

The principal aims of this implementation are

(a)   to demonstrate the feasibility and effectiveness of the AI based

approach to HEN synthesis task described in section 3.2, and

(b)   to evaluate the performance of the control strategy proposed in

section 3.2.4.

One of the most noteworthy developments in the area of knowledge

programming techniques has been the use of object-oriented programming

paradigm.   Increasing number of AI systems are being developed in this

paradigm.   Objects   provide   the highest degree of data abstraction and

encapsulation that leads to systems with excellent modularity.   This

feature makes the paradigm ideally suited for rapid prototyping and

exploratory programming.   Additionally, it permits hierarchic   and

non-hierarchic   inheritance of structure (variables and data) and

behavior (procedures that manipulate the data) among objects.   This

feature is highly desirable for representing and manipulating the

structural and taxonomic information required to effectively reason

about engineering domains such as HEN synthesis.   Based on these

criteria as well as the availability of a system, the LOOPS environment

[Bobrow and Stefik, 1983] on a Xerox AI workstation has been chosen as

the implementation medium for the present work.   LOOPS is built on top

of the powerful Interlisp-D [Xerox, 1982] environment, which is host to

the Xerox AI workstation.   It, therefore, extends the full power of the

system development and debugging tools of the Interlisp-D environment.

Additionally, it offers bit-mapped graphics with a   user-friendly

49

interface consisting of windows, menus and mouse. Combination of these features makes LOOPS one of the most powerful knowledge programming environment available. To have a better understanding of the structure of the prototype developed in the present work, an overview of LOOPS environment is essential.

4.1 AN OVERVIEW OF LOOPS

LOOPS, which stands for Lisp Object Oriented Programming System, integrates four programming paradigms; in addition to the conventional procedure-oriented paradigm, it offers object-oriented, data-oriented and rule-oriented paradigms. Since different programming paradigms provide different ways of representing and manipulating knowledge, for a given application some paradigms can be more cost-effective than others; the cost includes the resources required for developing, debugging and modifying a system. By allowing for choice and combination of paradigms, LOOPS enables us to build cost efficient application systems.

Out of the four paradigms offered by LOOPS, the present implementation employs predominantly the object-oriented paradigm. The procedural and data-oriented paradigms are sparsely used, and this sparse usage too, is within the framework of the object-oriented paradigm. The rule-oriented paradigm has not been used at all in the present implementation. In addition, the procedural and rule-oriented paradigms are more commonly found in programming languages than the other two. In the light of these facts, the discussion of LOOPS

50

features in this section has been restricted to the object and data-oriented paradigms only.

The procedure-oriented paradigm has been, by far, the most widely employed paradigm. In this paradigm, a program consists of a set of procedures (also called subroutines or functions in some languages). Data are kept separate from procedures that manipulate them. Large procedures are built from the small ones through the use of a composition mechanism: invocation of procedures through procedure calls. The procedural part of LOOPS is Interlisp-D [Teitelman, 1978; XEROX, 1982]; it is an enhanced version of Lisp with several data abstraction facilities and control structures added to the standard list processing features of Lisp. These enhancements include data structures, such as arrays, records, property lists, windows and menus, and control structures, such as decisions, case statements and complex iteration constructs, all in Pascal-like syntax. This paradigm is the foundation on which the rest of LOOPS is built.

The object-oriented paradigm in LOOPS derives its roots from Smalltalk [Ingalls, 1978; Goldberg, 1981; Goldberg and Robson, 1983] and Flavors [Weinreb and Moon, 1981; Cannon, 1982]. In this paradigm, a program consists of a set of objects combining both, data (called variables in LOOPS) and instructions (procedures) that manipulate the data (called methods in LOOPS). Larger objects are built up from the smaller ones by employing the composition mechanisms, which for this paradigm are specialization, hierarchical and non-hierarchical (multiple) inheritance, composite objects and perspectives.

51

Objects in an object-oriented paradigm are organized into object classes. An object class (or simply a class) is a description of one or more similar objects, each of which is termed an *Instance* of the class. Every object in LOOPS is an instance of exactly one class. Even classes themselves are instances of a class, usually the one called *Class*. Associated with each object class are its data (variables) and procedures (methods). Variables of a LOOPS object are classified into two categories: class variables and instance variables. Class variables are used to contain information shared by all instances of the class, i.e., the information pertaining to the class taken as a whole. Instance variables contain the information specific to an instance. Both kinds of variables have names, values and associated property lists. A class describes the structure of its instances by specifying the names and default values of instance variables. Unlike some other knowledge programming systems, e.g., KEE [Kehler and Fikes, 1985], LOOPS does not associate any data type with the object variables. In this regard, it retains the flexibility of LISP rather than opting for datatype rigidity of, say, Pascal.

All actions in an object-oriented programming come from sending messages to objects. Message sending is a form of indirect procedure call: instead of naming a procedure to perform an operation on an object, a message is sent to the object, which responds to the message by activating the appropriate method ("known" only to itself). A selector in the message specifies the procedure (*method* in LOOPS) that needs to be activated. A class associates selectors (Lisp atoms) with methods which are the Interlisp functions. All instances of a class use

52

the same selectors and methods. Any difference in the response by two instances of the same class is determined by a difference in the value of their instance variables.

A message in LOOPS has the following form;

$$(\leftarrow \textit{object} \quad \textit{selector} \quad \textit{arg1} \quad \textit{arg2} \; ...)$$

where ← is a short form of *SendMessage* command, *Object* is the recipient of the message (a class or an instance, depending on the *Selector*), *Selector* is a Lisp atom that specifies the Interlisp function to be invoked and *arg1, arg2,* etc. are optional arguments that are passed to the function. The effect of a message can be a change in the data values of one or more objects, or additional messages to one or more objects. A message returns the result of the last computation step to the sender. In this respect, it behaves like a function call in Lisp. Message sending supports the important concepts of data abstraction and encapsulation. Thus, an object need not know the internal data structures and the implementational details for the methods of other objects in order to communicate with them. Also, these details can be changed without affecting the inter-object communication.

Messages are usually designed in sets to define a uniform interface to all objects that support a specific operation. Such a set of related messages is called a protocol. When protocols are standardized, different classes of objects sharing these protocols can be treated uniformly. In fact, the object-oriented paradigm is particularly well suited to applications where the description of entities (in the form of object classes) is simplified by the use of uniform protocols. For example, in a graphics application, windows, lines and composite

53

structures could be represented as different object classes all of whose instances respond to a uniform set of messages (a standardized protocol) such as Display, Move and Erase. Such protocols extend the motion of modularity (interchangeable and modifiable pieces as enabled by message sending).

In LOOPS, object classes are organized in the form of an inheritance network, called lattice, with arcs determining the inheritance path. Inheritance supports the concept of specialization: the class at the destination of an arc (called subclass) is a specialization of the class at the source node (called superclass). All descriptions (instance variables, class variables and methods) of a class are inherited by its subclass(es). The fact that LOOPS forms a graph or lattice of classes and not a tree (as in Smalltalk) implies that it permits multiple inheritance. In other words, not only a class can have several subclasses, but it can also have several superclasses. These two forms of inheritance, hierarchical (single superclass) and non-hierarchical (multiple super classes), reduce the need to specify redundant information and simplifies updating and modification, since information can be edited and changed at one place. Changes to the inheritance network are very common during program development; new classes are created and existing ones are reorganized. The LOOPS environment facilitates such changes with the help of an interactive graphics package called browser for adding and deleting classes, renaming classes, splitting and specializing classes, rerouting inheritance paths in the lattice, adding deleting or modifying the variables and methods and so on.

54

In the data-oriented programming, action is potentially triggered when data are accessed. For this reason, this paradigm is often referred to as access-oriented paradigm. Its basic mechanism in LOOPS is a structure called active value, which enables a programmer to specify whether any special procedure is to be invoked on read or write access to a variable of an object. LOOPS checks on every variable access whether the value (or the property being accessed) is marked as an active value. If it is, then the procedure specified by the active value will be executed. LOOPS employs the following convention for active values;

$$\#(localState \qquad getFn \qquad putFn)$$

The *localState* is a place for storing data. The *getFn* is the name of an Interlisp function invoked whenever a read access ("get" operation in LOOPS) is made to the data value. Similarly, the *putFn* is the name of an Interlisp function invoked whenever a write access ("put" operation in LOOPS) is made to the data value. Every active value need not specify both the functions; if any of the function name is NIL, then for the corresponding operation, the data value can be accessed normally, without any side effects. The *getFn* and *putFn* can be user defined or built-in functions provided by the system.

The mechanism of active values is dual to the notion of messages. A message is a way of telling objects to perform an operation, which can change their variable values as a side effect. Active value is a way of accessing variables, such that an operation is performed (e.g., a message is sent) as a side effect. Composition in this paradigm is carried out by nesting the active values, thus allowing a programmer to

specify multiple access functions for a variable. Note that LOOPS restricts the use of active values only to the object variables; it does not permit the data items of the procedural paradigm to use the active values. The active values can be thought of as probes that can be placed on the object variables of a LOOPS program. For example, active values drive gauges that display graphically the values of object variables; whenever the value of a variable changes, its graphical image gets changed appropriately. Nested active values are analogous to multiple probes. However, it is desirable that these "probes" are for independent purposes only and do not interfere with each other. Thus this paradigm is most suitable for programs that monitor other programs.

LOOPS derives its strength not by merely putting together different paradigms, but by integrating them to such an extent that they almost loose their individual identities. The paradigms, therefore, not only complement each other, but also work together as one single environment. The following examples illustrate the integration of paradigms in LOOPS:

(a) Methods in object-classes are Interlisp functions.

(b) The procedures in active values can be Interlisp functions, or calls on methods (messages).

(c) Variables of an object can be active values.

Many of the facilities of the Interlisp-D environment are extended to this integrated LOOPS environments. These facilities include the display-oriented break package, editors and inspectors, windows and menus, DWIM (Do What I Mean-a spelling correction facility), Programmer's Assistant (a package that acts as intelligent intermediary between the user and system), and Masterscope (an interactive package

56

for analyzing and cross referencing user programs). Thus the integrated paradigms in LOOPS not only benefit the structure and performance of the application systems, but also facilitate the process of developing, maintaining and modifying these systems.

4.2 STRUCTURE OF HENSYN

The search system for the heat exchanger network synthesis described in section 3.2, has been implemented on a XEROX 1108 AI workstation with the hybrid knowledge programming environment LOOPS. The system, called HENSYN, has been developed entirely within the framework offered by the object-oriented paradigm of LOOPS; sparse use has been made of the procedural and access-oriented paradigms.

For each element of the problem domain, such as process stream, match, grid diagram, match matrix, etc., there is a corresponding object class in HENSYN. This feature makes it easier to model the HEN synthesis process in the LOOPS environment. However, this same feature renders it very difficult to describe unambiguously the features and behavior of the system. To prevent possible confusion, the following notational scheme has been adapted in the remainder of this chapter. The real-world domain elements (entities) are described in normal text words, whereas the corresponding objects classes in the system are preceeded by a $ sign. The instances of these object classes are indicated by a prefix #$ followed by the identifier for the instance. The method (procedure) and variable names are italicized. Additionally, all LOOPS identifiers (object classes, instances, methods and variables)

begin with a capital letter to differentiate from the real-world entities which are entirely in the small case letters. Finally, the "owner" instance of a method or a variable is referred to as Self for brevity.

Figure 4-1 shows the object world of HENSYN. Note the use of hierarchical inheritance to specialize the class $Stream to subclasses $ProcessStream and $Utility; these classes are, in turn, specialized into the corresponding hot and cold streams. Obviously, with these specializations, there will be no need for instances of the two superclasses, $ProcessStream, $Utility and $Stream. For each HEN synthesis problem, the following object instances are required:

(a) One instance each of classes $Problem, $MatchMatrix and $GridDiagram.

(b) A set of specialized stream instances (of classes $HotUtility, $ColdUtility, $HotStream and $ColdStream).

(c) A set of instances of $Match; one for each pair of stream instances, except when both are utilities.

The number of specialized streams, and consequently, the number of $Match instances, are determined by the problem statement.

Inter-object communication has been simplified by the use of uniform protocols. For example, all four classes, $HotUtility, $ColdUtility, $HotStream, and $ColdStream, respond to a common message *IncreaseQ*, albeit with different effects. Thus, any other object, say an instance of $Match, need not worry about which type of object is at the receiving end of the message *IncreaseQ*. This uniformity enables
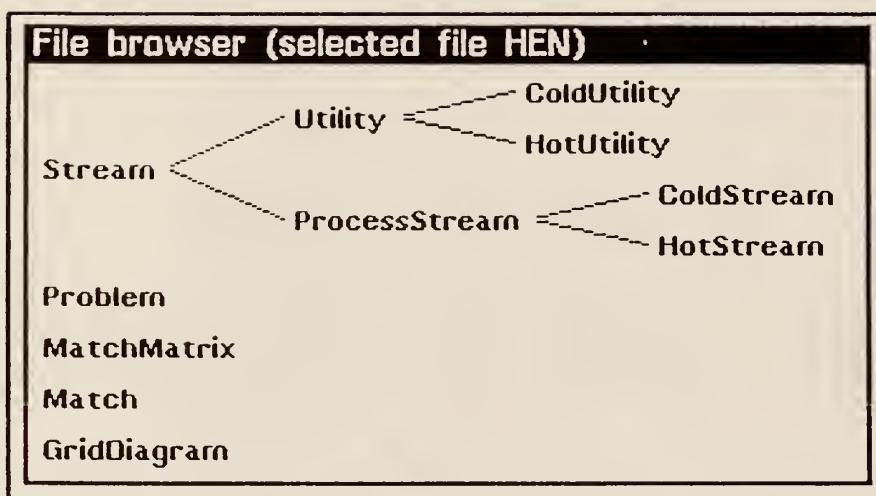
Fig. 4-1.   The object world of HENSYN.

other objects to treat these four classes of objects in identical fashion, thus considerably simplifying the system design.

4.2.1 State Space

At any instant, the state of the problem is defined by the instances of classes $HotUtility, $ColdUtility, $HotStream, $ColdStream and $Match. Note that the first two are the specialization of $Utility, and the next two are the specialization of $ProcessStream. Together, these four are specialization of class $Stream, and therefore, will be referred to as specialized streams. The information defining the state of the problem is stored as the values of variables of these object instances.

Both, $HotUtility and $ColdUtility have an identical structure, inherited from the superclass $Utility. Each instance has three instance variables (IVs): the initial heat duty of the stream as specified by the problem data (*Q*), the unsatisfied or remaining heat duty (*CurrentQ*) and a list of matches ($Match instance identifiers) into which it participates (*AllMatches*). The value of *CurrentQ* is set to that of *Q* at the time of first usage with the help of the active value facility. The initial value of variable *Q* is initialized to the value specified in the problem. Once again, the initialization is done through the active value mechanism. The value of *AllMatches* is NIL to begin with (default value inherited by all instances) and is updated as and when a $Match instance is created for that stream. Neither $HotUtility nor $ColdUtility has any class variables.

Classes $HotStream and $ColdStream also have identical structures, inherited from the superclass $ProcessStream, consisting of the following instance variables (IVs).

(a) The three characteristic values of a stream at the start of the synthesis, viz., the specific heat flow rate (*mcp*), the initial source temperature (*SourceTemp*), and the initial target temperature (*TargetTemp*). All three have default value NIL and are initialized to the values specified by the problem.

(b) Present values of the source and target temperatures (*CurrentSourceTemp* and *CurrentTargetTemp*, respectively), set equal to *SourceTemp* and *TargetTemp*, respectively, at the time of first usage.

(c) A list of all matches ($Match instance identifiers) in which the stream participates (*AllMatches*). Once again its default value is NIL and is updated whenever a $Match instance involving this stream is created.

(d) The initial heat duty of the stream (*Q*), as specified by the problem. This value is computed from the values of *mcp*, *SourceTemp* and *TargetTemp* at the time of first usage.

(e) Unsatisfied heat duty of the stream (*CurrentQ*), set equal to Q at the time of first usage. This value is modified as matches involving this stream are made.

Note that initializations in (b), (d) and (e) above are done through the active value facility of LOOPS.

Each instance of class $Match contains the information on the actual or potential match between a pair of streams, one is a hot stream

61

(an instance of $HotUtility or $HotStream) and the other, a cold stream (an instance of $ColdUtility or $ColdStream). A $Match instance contains the following instance variables:

(a) the identity of the hot and the cold streams that constitute the match (h and c, respectively),

(b) the match load as per the elimination strategy (Q),

(c) the quantity $\left\{ \dfrac{1}{(mc_p)_h} - \dfrac{1}{(mc_p)_c} \right\}$ (cf. Equations 2.5 and 2.8) for calculating the feasibilities (*mcpFactor*),

(d) the feasibilities for hot and cold end matching (*HEMfeasibility* and *CEMfeasibility*, respectively), and

(e) the status of the match (*Status*).

The default values of h, c and *mcpFactor* are NIL. Their actual values are set during initialization and do not change subsequently. The default values of Q, *HEMfeasibility* and *CEMfeasibility* are 0 (zero); their values are set (reset) whenever the instance is "asked" to recompute its feasibility. The default value for *Status* is Open indicating that the instance is available for selection of the next match. The value of *Status* changes from Open to HTU on selection of the instance as one of the matches in the solution, and from Open to Close when one (or both) of the two streams constituting the match, h and c, are eliminated.

4.2.2 Operators

The operators for the search system, introduced in section 3.2.2,
have been implemented as a set of procedures (called methods in LOOPS)
distributed over all object classes. At present, only two operators
have been implemented: Match and Unmatch.   The remaining two, Split and
Merge, can be implemented in a similar fashion.   Figures 4-3 and 4-4
show the sequences of methods invoked in order to apply the operators
MATCH and UNMATCH, respectively, to a LOOPS configuration of a problem
state.   In these figures, the boxes with names represent object classes.
A set of names underneath each box is the list of methods participating
in the chain of events.   Each unbroken directed line represents a
message sent by one object to the other.   Such a line originates at the
method that sends the message and terminates at the method being
invoked.   Each line is numbered to portray the order of the messages.   A
brief description of the events that take place for each operator
follows.

MATCH: When the control strategy determines that this operator is to be
applied, a message *MakeMatch* is sent to the $MatchMatrix instance,
thereby causing the invocation of method *MatchMatrix.MakeMatch*.   This
method sends two messages.   The first one, *GetNextMatch* (message # 1 in
Figure 4-2), is to Self (the $MatchMatrix instance) for selecting a pair
of streams and the location of match (hot or cold end).   Note that this
method is a part of the control strategy, since the manner in which
these choices are made depends upon the control strategy.   Having

63
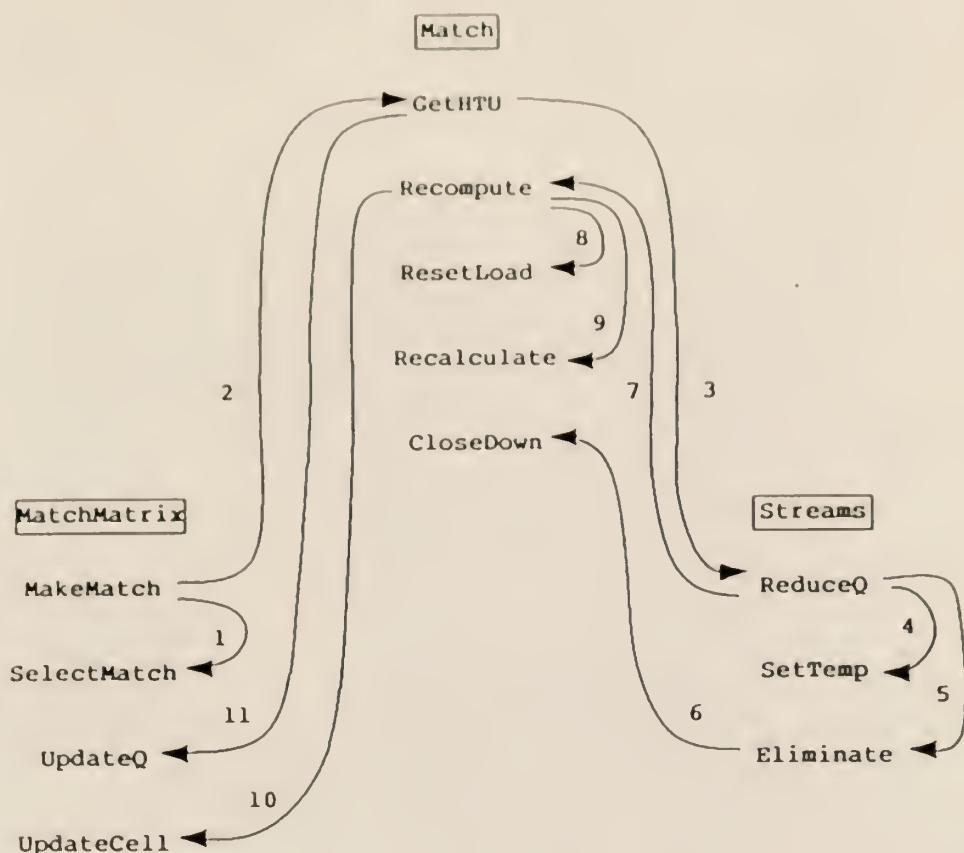
Fig. 4-2.   Implementation of MATCH operator.

obtained the identity of the match (i.e., the pair of streams to be matched) and the location of match, the second message, *GetHTU* (message # 2), is sent to the appropriate $Match instance. This message "asks" the $Match instance to obtain an HTU for itself. The location of match, hot or cold end, is passed to the instance whereas the extent of match is "known" to the instance, as instance variable Q.

In response to message *GetHTU*, the $Match instance changes its status from Open to HTU, sends message *ReduceQ* (# 3) to each of its "parent" or constituent streams (known to it as IVs h and c), and finally, asks the match matrix to update the entries (values being displayed at that moment) for the heat duties of the parent streams (message # 11, *UpdateQ*).

The parent streams (one hot and one cold), on receiving the message *ReduceQ*, reduce their remaining heat duties (IV *CurrentQ*) by the amount specified in the message. Each of them then checks to see if it has been eliminated, i.e., to see whether or not the remaining heat duty is zero within a tolerance of 0.5. If the stream has been eliminated, a message *CloseDown* (# 6) is sent to each of the matches involving this stream to change its status from Open to Close; if *Status* has a value Close or HTU, then the message is ignored. If the stream has not been eliminated, instead of message *CloseDown*, message *Recompute* (# 7) is sent to each of the matches involving this stream. In either case, if the stream is a process stream, its temperature at the location of the match (hot end or cold end) is adjusted appropriately by sending message *SetTemp* (# 4) to itself.

Each match ($Match instance), receiving message *Recompute*, resets its load as per the elimination strategy (message *ResetLoad*, # 8) and recalculates its feasibilities (message # 9, *Recalculate*) at the two ends using the elimination conditions (cf. Equations 2-4 through 2-9). Finally, message *UpdateCell* (# 10) is sent to the match matrix for updating the values being displayed for this match.

UNMATCH: When the control strategy determines that this operator is to be applied, message *Unmatch* is sent to the match matrix, thereby causing the invocation of method *MatchMatrix.Unmatch*. This method "undoes" the effect of the latest application of the MATCH operator by sending message *ReleaseHTU* (# 1 in Figure 4-3) to the appropriate $Match instance. (The sequence of all previous MATCH operator applications is available as IV *PastMatches* of the $MatchMatrix instance). This instance of $Match sends messages *IncreaseQ* (# 2) to its parent or constituent streams, changes its status from HTU to Open, and informs the match matrix to update the values displayed for the remaining heat duties of its parent streams (message # 9, *UpdateQ*).

The parent streams, one hot and one cold, receiving the message *IncreaseQ*, increase their remaining heat duties (values of IV *CurrentQ*) by the amount specified in the message. Each stream then checks to see whether or not it has been eliminated as a result of the match being undone. If so, then it undoes the effect of elimination by sending message *OpenUp* (# 4) to each of the instances involving itself; otherwise message *Recompute* (# 5) is sent to each of the matches involving this stream. In both cases, the temperature of the stream, if

66
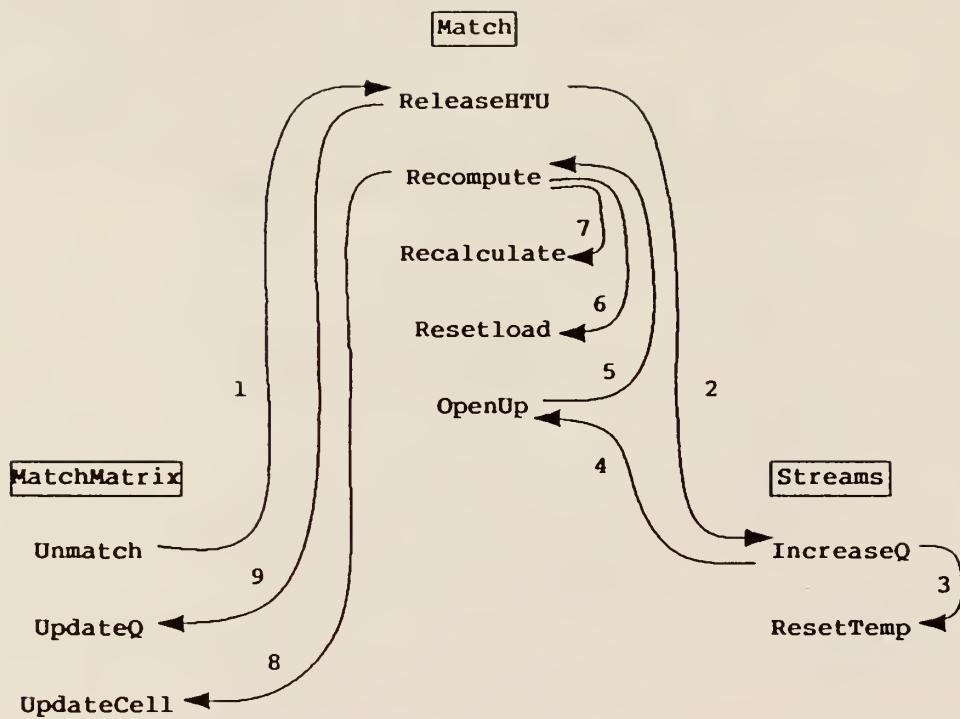
Fig. 4-3.   Implementation of UNMATCH operator.

it is a process stream, at the end where the match is being undone, is reset by sending message *SetTemp* (# 3) to Self.

Each of the matches receiving the *Recompute* message responds in the same manner as in the case of Match operator: it resets its load as per the elimination strategy (message # 6, *ResetLoad*), recalculate the feasibilities at the two ends (message # 7, *Recalculate*), and informs the match matrix of the changes by sending message *UpdateCell* (# 8).

4.2.3 Control Strategy

The control strategy described in section 3.2.4 has been implemented in an interactive, menu-driven form. The user chooses the operator to be applied to a problem state through a permanent menu attached to the match matrix. Since only the MATCH and UNMATCH operators have been implemented, the choice is limited to the corresponding options, MakeMatch and Unmatch in the menu. Figure 4-4 displays the sequence of messages sent and the corresponding methods that are invoked for the operation of the control strategy in HENSYN. If the MATCH operator is chosen, message *MakeMatch* is sent to the $MatchMatrix instance. This method, in turn, invokes the method *GetNextMatch* (message # 1 in Figure 4-4) to obtain the match selection for applying the operator MATCH. A match selection, *sel*, as shown below, consists of the identity of the $Match instance and the location of the match (hot or cold end).

sel : (*matchID* *location*)

68
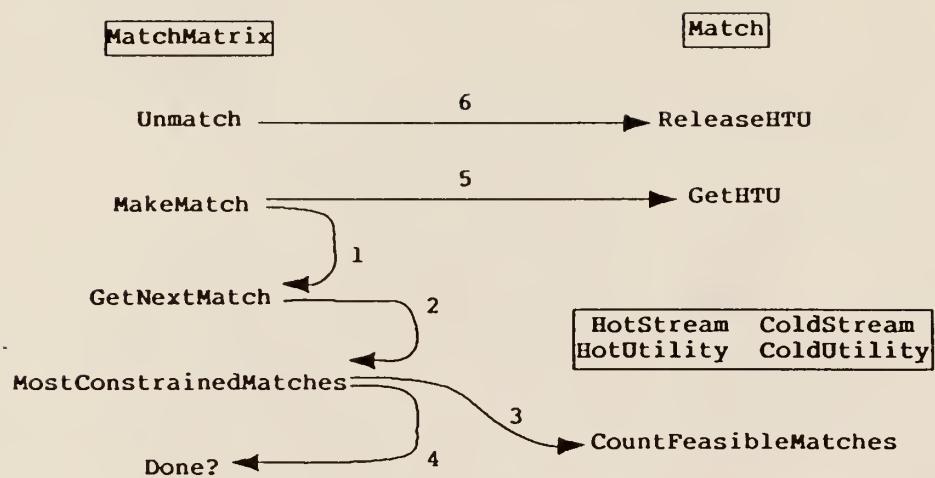
Fig. 4-4. Implementation of control strategy.

Having obtained the $Match instance and the location of the match, message *GetHTU* (# 5) is sent to this instance, with the location as an argument.

Method *GetNextMatch* maintains and updates the history of available alternate selections for all previously "visited" states, in IV *Alternates* of the $MatchMatrix instance, having the following form.

Alternates : *((state1   sel1   sel2 . . .)*

(state2   sel1   sel2 . . .)

. . .   . . .))

where *state1*, *state2*, etc. are the unique identifications for the problem states, in the form of a list of matches (HTUs) present in the partial network corresponding to the problem state (the value of IV *PastMatches* of $MatchMatrix instance), and *sel1*, *sel2*, etc. are the unexplored alternate selections ($Match instance and location pairs as described above) for the corresponding problem state. When invoked the method first checks if the present state has been "visited" before. If it has been, the corresponding alternate selections from IV *Alternates* constitute the list of most constrained matches from which one selection is chosen for applying the MATCH operator and returned to the method sending the message (*GetNextMatch*); if not, message *MostConstrainedMatches* (# 2) is sent to Self, which returns the list of the most constrained matches for making a selection.

If the list of most constrained matches, obtained in either of the above two cases, is empty (NIL), the user is informed of the situation and the system suggests the use of UNMATCH operator for backtracking to obtain alternate match choices. If only one alternate selection is

70

available in the list of the most constrained matches, this selection is returned to method *MakeMatch*. If more than one alternate choices are available, the user is asked to select one of these matches through a pop-up menu. If the selected match is feasible only at one end, this end is selected; otherwise, the user is asked to make a selection from another pop-up menu having two options, hot and cold. The selected match-location pair is removed from the list of the most constrained matches for the present state and the value of IV *Alternates* is updated accordingly.

Method *MostConstrainedMatches* sends message *CountFeasibleMatches* (# 3) to each of the uneliminated streams (i.e., the streams having non-zero heat duty) to determine the list of matches it can make and the number of ways in which it can make these matches (called number of matching possibilities). For the latter, a match at the hot end and the same match at the cold end are counted as two matching possibilities. Next, the method *MostConstrainedMatches* determines the streams that have the lowest non-zero number of matching possibilities. If there are no such streams, then either a solution state or a dead-end state has been reached. To check this, message *Done?* (# 4) is sent to Self, the rest of the steps are skipped and the application of MATCH operator is aborted. When one or more streams having the least number of matching possibilities are obtained, for each match of each of these streams, the number of matching possibilities of the other stream (participating in the match) is examined and the matches with the lowest such number are collected, along with the location(s) at which the streams can be

matched. A list of these match-location pairs is returned to the sender of the message (method *GetNextMatch*) as the most constrained matches.

Method *Done?* examines the remaining or unsatisfied heat duties (IV *CurrentQ*) of every stream in the problem. If all are zero, a solution state has been reached; otherwise a dead-end state is reached. The method informs the user about its findings and suggests that alternate solutions can be obtained by backtracking with the help of the UNMATCH operator.

Method *CountFeasibleMatches* of class $Stream, which is inherited by the instances of $HotStream, $ColdStream, $HotUtility and $ColdUtility, examines the feasibilities (IV *HEMfeasibility* and *CEMfeasibility* of the $Match instance) of each of its matches (stored in IV *AllMatches* of the corresponding stream instance) and makes a list of feasible matches (IV *FeasibleMatches*) and counts the total number of ways in which the stream can be matched (IV *MatchingPossibilities*) with the hot and cold end matches counted separately. No value is returned to the message sender; the values of the two IVs set by this method are used subsequently, by method *MostConstrainedMatches*.

If UNMATCH is the operator selected by the user, message *Unmatch* is sent to the match matrix. Since only the last match can be undone, the user is not asked to specify the match. The system keeps track of the sequence of past matches including the locations (as IV *PastMatches*), from which it selects the latest one and supplies it to operator UNMATCH. The $Match instance thus selected is sent message *ReleaseHTU* (# 6 in Figure 4-4) with location as the argument.

72

With this interface, the control strategy described in section 3.2.4 can be executed very easily in an interactive fashion. The user is required to carry out the first part of the control strategy, viz. the depth-first search among the available choices for the most constrained matches determined by the system. The system keeps track of the alternate unexplored matches for each "previous" state and determines the most constrained matches for a "new" state.

4.2.4 User Interface

The HENSYN system provides a very friendly user interface, which forms a substantial chunk of the system. This interface performs the following major tasks:

(a) Obtains the problem specifications (data) from the user.

(b) Instantiates and initializes the objects required for a problem.

(c) Displays the current state of the problem in the form of a match matrix.

(d) Provides an interactive framework for executing a control strategy.

(e) Displays the network design in the form of grid diagram.

Additionally, it provides several utility functions for the user's convenience.

The interface consists of three objects, each of which is an instance of different object classes: $Problem, $MatchMatrix, and $GridDiagram. Tasks (a) and (b) are performed by the $Problem instance,

tasks (c) and (d) by the $MatchMatrix instance, and task (e) by the $GridDiagram instance.

To use the system for solving a HEN problem, the user invokes the Interlisp function HENSYN by typing the following in the Interlisp-D executive window:

(HENSYN)

The system in response, pops up a menu containing the list of problems for which it has the data. Additionally, an option "*NewProblem*" is available for solving a problem that is not present in the menu. If the user selects the "*NewProblem*" option, the system asks the user for the name of the problem, creates a new instance of class $Problem, and sends message *Initialize* to this instance. If the user selects one of the "available" problems, message *Reinitialize* is sent to the corresponding $Problem instance. The only difference between the messages *Initialize* and *Reinitialize* is that the former obtains the data from the user (steps 1 and 2 below), where as the latter skips this part, since the data is already stored in the corresponding $Problem instance. Each of these messages starts a chain of events that creates and initializes the necessary objects and sets up the match matrix and grid diagram. This chain of events is shown in Figure 4-5. This diagram is identical in nature to Figures 4-2, 4-3 and 4-4. The only additional element in this diagram is a dashed line, which represents the creation of one or more new instances of an object. It originates at the method that carries out the instantiation and terminates at the box corresponding to the object (class) being instantiated. This line too is numbered to indicate the order of sending messages. The following is a detailed

74
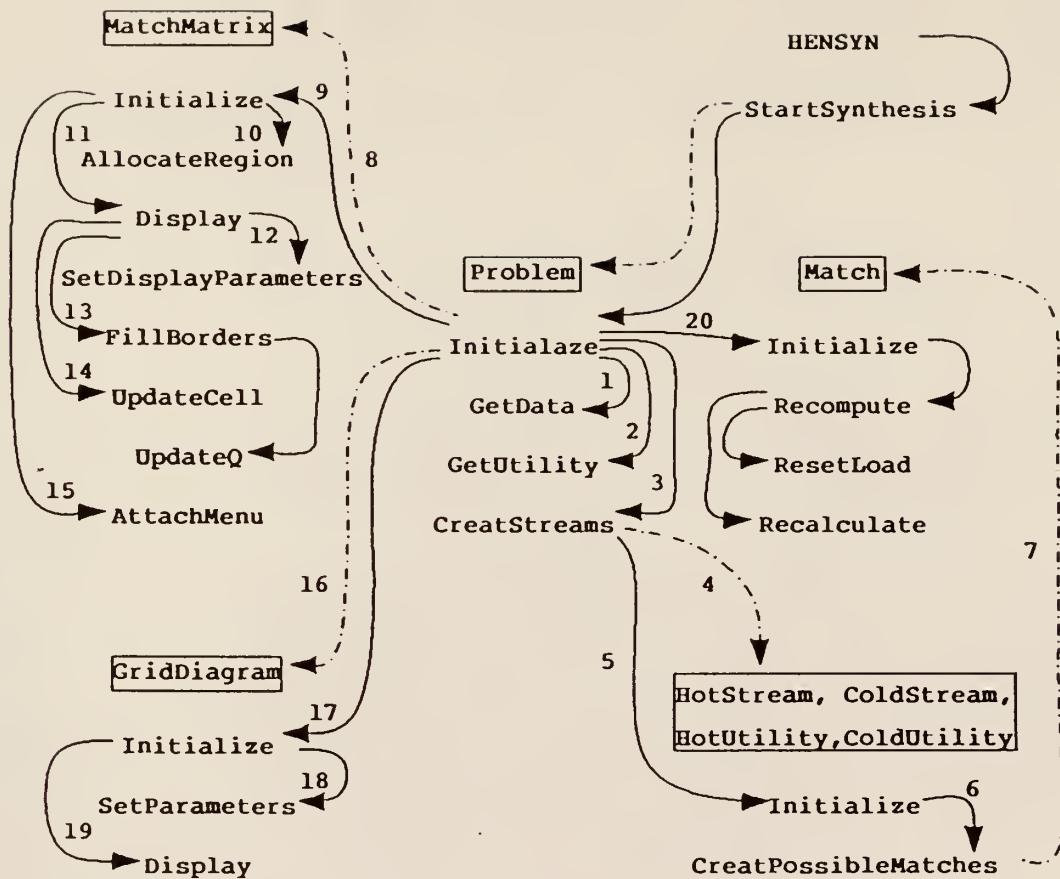
Fig. 4-5. Initialization of HENSYN: chain of events.

description of the events depicted in Figure 4-5. Note that the event numbers in this description correspond to the message lin numbers in Figure 4-5.

(1) The user is asked for the problem data. The system prompts the user to supply the name (ID tag), specific heat flow rate, source temperature and target temperature of a process stream. This procedure is repeated until the user types "none" for the name of the next stream. All this information is stored as a list of records in instance variable *DataTable* of the $Problem instance.

(2) The user is asked for the minimum approach temperature, stored as IV *TDmin*, and the amounts of hot and cold utilities along with their names (ID tags for the corresponding instances of $HotUtility and $ColdUtility), which are stored as a list of individual records in IV *Utilities*. Multiple streams are allowed for hot and cold utilities. The end of each type of utility is indicated by typing a zero for the amount.

(3) The $Problem instance sends a message to itself for instantiating and initializing the streams.

(4) Instances of $HotStream, $ColdStream, $HotUtility, and $ColdUtility are created, based on the values of IVs *DataTable* and *Utilities*.

(5) The stream instances created in (4) are "asked" to initialize themselves. Each utility sets its amount (IV *Q*) and each process stream sets values for its source temperature, target temperature and specific heat flow rate (IVs *SourceTemp*, *TargetTemp* and *mcp*,

76

respectively). These values are supplied to the instances as arguments of the message *Initialize*.

(6)   As part of initialization, each instance sends message *CreatePossibleMatches* to itself for creating all possible candidate matches with the opposite type of streams. Thus, an instance of $HotStream will create $Matches instances for each of the existing $ColdStream instances as well as $ColdUtility instances and an instance of $ColdStream will create $Match instances for each of the existing $HotStream instances as well as $HotUtility instances. $HotUtility and $ColdUtility instances create $Match instances for the existing $ColdStream and $HotStream instances, respectively. No utility creates a $Match instance for the opposite type of utility instances.

(7) $Match instances are created as described in (6) for each pair of streams, one hot and one cold, except when both are utilities.

(8)   Having instantiated all the streams and matches, the method *Initialize* of $Problem now creates an instance of $Match Matrix. This instance is assigned a name MM and its identity is stored in the IV *MMid* of the $Problem instance.

(9)   The newly created instance of $MatchMatrix, #$MM, is asked to initialize itself. Method *Initialize* of $MatchMatrix assigns column numbers to the hot streams, including the hot utilities, if any, and row numbers to the cold streams, including the cold utilities, if any. These assignment lists are stored in IVs *HotStreamList* and *ColdStreamList*, respectively, to be used in displaying the match matrix. Next, each $Match instance is

77

assigned a cell in the match matrix. The assignment list of all matches and their cell regions are stored in IV *MatchList*. Finally, the identity of the $MatchMatrix instance is stored as class variable (CV) *MatchMatrixID* of class $Match. This communication link is essential for matches to send update messages to the match matrix.

(10) For each $Match instance, the match matrix asks itself to allocate a cell region in the match matrix window, based on the row and column numbers of the constituent or parent streams.

(11) Having completed the initialization, the match matrix is asked to display itself. It does so in four steps: first, message *SetDisplayParameters* is sent to Self for setting the display parameters. Next, a window is opened for displaying match matrix and this window is divided into cells by drawing horizontal and vertical lines. The cell size is fixed and the window size is determined based on the number of rows and columns required for a problem. The third step is to fill all the border cells, including the first and last rows, and the first and last columns; this is accomplished by sending message *FillBorders* to Self. The last step is to fill the non-border cells; this is carried out by sending message *UpdateCell* to all the $Match instances.

(12) Based on the number of streams, four display parameters are set: the number of rows and columns in the match matrix (IVs *rows* and *Columns* of $MatchMatrix, respectively), and the height and width of the match matrix (IVs *Height* and *Width*, respectively).

78

(13)   Method *FillBorder* of the $MatchMatrix instance fills the first

and last row as well as the first and last columns of the "empty"

match matrix. The first row contains the labels of the hot

streams, which are displayed by accessing IV *Label* of the

corresponding stream instances, and a label Qc in the last cell.

The first column, containing the labels of cold streams and a Qh

in the last cell, is displayed in a similar fashion. The

remaining cells of the last row and the last column contain the

unsatisfied heat duties of hot and cold streams, respectively;

these are displayed by sending message *UpdateQ* to Self.

(14)   Method *UpdateCell* locates the cell corresponding to the specified

$Match instance, erases the current contents of the cell and

writes the new information for this match. This information

depends on the status of the match. If the status is Open, then

the feasibilities of hot and cold end matching is displayed: an

"H" if the match is feasible at the hot end, and a "*" otherwise;

a "C" if the match is feasible at the cold end, and a "*"

otherwise. If the status is HTU then the heat load of the HTU is

displayed (IV *Q* of the corresponding $$Match instance). If the

status is Closed, "———-" is displayed.

(15)   The $MatchMatrix instance creates a synthesis menu and attaches

it to the match matrix displayed on the screen. At present, this

menu has five options: MakeMatch and Unmatch corresponding,

respectively, to the two operators MATCH and UNMATCH, Reset to

undo all the matches to restart the synthesis, ReDisplay to erase

and redraw the match matrix and grid diagrams, and Quit to stop

synthesis and delete the synthesis menu. After the initialization, the user interacts with the system solely through this menu.

(16) A new instance of $GridDiagram is created. Its identity is stored in the $Problem instance (as IV *GDid*) and in the $MatchMatrix instance (as IV *Grid*).

(17) The newly created instance of $GridDiagram is "asked" to initialize itself. As part of this initialization, a window is created for displaying the grid diagram for the current problem. The vertical distance between two adjacent streams and the horizontal spacing between two adjacent HTUs are fixed. The size of the window is determined based on these spacings and the number of streams in a problem. Next, a message *(SetParameters)* is sent to Self for determining the values of display parameters. Finally, the window is "filled in" by sending a message *Display* to Self.

(18) Method *SetParameters* of $GridDiagram determines the vertical position of each stream on the grid diagram. The lists of tuples (stream ID, vertical position) are stored in IV *HotStreams* for the instances of $HotStream and in IV *ColdStreams* for the instances of $ColdStream.

(19) Method Display draws the initial grid diagram in the window created earlier by the method *Initialize* of $GridDiagram. For each hot or cold process stream, a directed line is drawn, with arrowhead at the appropriate end, and the information pertaining to the right and left edges of the line are displayed. The right

80

edge contains the label of the stream, specific heat flow rate (parenthesized) and cold end temperature of the stream (target temperature for a hot stream and source temperature for a cold stream). The left edge contains the hot end temperature (source temperature for a hot stream and target temperature for a cold stream), the unsatisfied or current heat duty and the label of the stream.

(20) The last step in initializing a problem is to initialize all the $Match instances for the problem. Method *Initialize* of $Match sets the values of IV *mcpFactor* if both constituent streams are process streams, and then sends message *Recompute* to itself for determining the feasibilities at the two ends. Method *Recompute* determines the heat load of the resultant HTU and computes the values of IVs *HEMfeasilibity* and *CEMfeasibility*, all by resorting to the elimination strategy.

The initialization is now complete and the user can generate possible network solutions for the HEN problem by interactively executing the control strategy (as described in section 3.2.3) through the synthesis command menu attached to the match matrix. This menu has five options:

(1) <u>MakeMatch</u>: Initiates the application of MATCH operator to the present state of the problem by sending message *MakeMatch* to the $MatchMatrix instance.

(2) <u>Unmatch</u>: Initiates the application of UNMATCH operator to the present state of the problem by sending message *Unmatch* to the $MatchMatrix instance.

(3) ReDisplay: Clears the grid diagram and the match matrix windows and redraws/rewrites the contents.

(4) Reset: Brings the problem back to the start state by unmatching all existing matches and clearing the history by setting the IV *Alternates* to NIL.

(5) Quit: Deletes the synthesis command menu and stops further synthesis.   Note that the $Problem instance, which contains the data for the problem remains in the system to be used subsequently when needed.

When additional operators SPLIT and MERGE are implemented, the synthesis command menu can be expanded to include these operators. This menu acts as the sole means of obtaining the user inputs during the process of HEN synthesis.   The system uses the prompt window to keep the user informed of its activities and to display its suggestions/findings during the HEN synthesis process.

4.3 PERFORMANCE ANALYSIS OF HENSYN

The implementation described in the previous section constitutes the first prototype of the knowledge-based system for HEN synthesis.   In these section, the performance of the HENSYN system is analyzed in the light of the aims delineated at the beginning of this chapter.   Towards this end, the succeeding subsections will

(a) show how the present system can be employed to generate a set of networks satisfying the desired optimality criteria,

(b)   evaluate the performance of the control strategy employed in   the
      present implementation, and

(c)   identify the possible modifications for enhancing the performance
      of the present system.

These tasks will be accomplished by solving four standard test problems
and   examining their results.   These test problems have been proposed by
various researchers over the last two decades and are widely used for
benchmarking purposes.   Two factors are usually considered for comparing
the results obtained by different systems: efficiency (how "fast" the
solution has been reached) and quality (how "good" the solution is).   As
discussed in section 2.2,   the   desired   quality   of   solution   for   the
present system has been predetermined to be a set of networks having the
minimum number of HTUs and the minimum utility consumption.   Due to the
domain   knowledge   (in   the form of elimination strategy) built into the
system, it only "looks" for the networks satisfying the two optimality
criteria.   Analyzing the performance therefore, reduces to examining how
many   candidate   networks   are   obtained   and   how   many   dead-ends   are
encountered in the process.

4.3.1 An Example of HENSYN Usage

    The   usage   of   the   system   is exemplified with the help of a test
problem,   the   so-called   6SP2   problem,   first   reported   by   Shah   and
Westerberg [1975].   The problem consists of six process streams, three
hot streams, H1, H2 and H3, and three cold streams, C1, C2 and C3.   The
minimum utility requirement is one hot utility stream, HU1, with a heat

83

duty of 1436.2 units. The characteristic values of all the streams are reproduced in Figure 4-6. The minimum driving force is 11.1 degrees.

To start the system, the user types *(HENSYN)* in the Interlisp-D executive window. The system responds with a menu containing the list of existing (previously solved) problems and an additional choice "*NewProblem*". Presuming that the system does not have the data for the 6SP2 problem, the option "*NewProblem*" is selected. The system prompts for the name of the problem to which the user types in *6SP2*. The system then starts asking the user for the data for this problem. Figure 4-7 shows the dialogue between the user and the system during initialization. For each stream, the user needs to input the label (stream ID), specific heat flow rate (mc_p), source temperature and the target temperature; these values constitute the information in the corresponding row in the data table of Figure 4-6. The end of data is indicated by typing "none" for the label (ID) of the next stream. The system then asks for the amounts of hot and cold utility requirements for the problem; for each utility the amount and label are asked for. The end of each type of utility streams is indicated by entering 0 (zero) for the amount of next stream. After obtaining the utilities, the system starts creating and initializing various object instances for the present problem (cf. section 4.2.4). Each time a new object is created or initialized, a message is displayed in the Interlisp-D executive window. Finally, the match matrix and grid diagram for the problem are displayed in two new windows; the user is asked to specify the location of these windows. The match matrix window has a synthesis

84

Stream ID	mc_P	Source Temp	Target Temp
H1	14.77	271.1	148.9
H2	7.17	198.9	82.2
H3	10.53	187.8	93.3
C1	8.07	37.8	226.7
C2	11.61	82.2	226.7
C3	18.71	60.0	160.0

$\Delta T_{min} = 11.1°$

Minimum Utility Requirement: hot utility (HU1): 1436.2 units.

Fig. 4-6. Data for 6SP2 problem.

```
NIL
65+(HENSYN)

  Name for new problem --> 6SP2
Please type in the data for all the streams
To indicate the end of data,
    type none for stream name
Name of the next stream (# 1) --> H1
Specific heat flow rate (mcp) for H1 --> 14.77
Source temperature for H1 --> 271.1
Target temperature for H1 --> 148.9

Name of the next stream (# 2) --> H2
Specific heat flow rate (mcp) for H2 --> 7.17
Source temperature for H2 --> 198.9
Target temperature for H2 --> 82.2

Name of the next stream (# 3) --> H3
Specific heat flow rate (mcp) for H3 --> 10.53
Source temperature for H3 --> 187.8
Target temperature for H3 --> 93.3

Name of the next stream (# 4) --> C1
Specific heat flow rate (mcp) for C1 --> 8.07
Source temperature for C1 --> 37.8
Target temperature for C1 --> 226.7

Name of the next stream (# 5) --> C2
Specific heat flow rate (mcp) for C2 --> 11.61
Source temperature for C2 --> 82.2
Target temperature for C2 --> 226.7

Name of the next stream (# 6) --> C3
Specific heat flow rate (mcp) for C3 --> 18.71
Source temperature for C3 --> 60.0
Target temperature for C3 --> 160.0

Name of the next stream (# 7) --> none

Minimum allowable driving force --> 11.1          .

Enter the hot utilities for 6SP2 problem ...
   (At end enter 0 for amount)

          Amount    --> 1436.2
          Name(ID) --> HU1

          Amount    --> 0

Enter the cold utilities for 6SP2 problem ...
   (At end enter 0 for amount)

          Amount    --> 0
```

Fig. 4-7.   Initialization of 6SP2 problem.
                             .

86

command menu attached to it. Figure 4-8 shows the screen image at the end of initialization.

At this time the user can proceed to synthesize a network by choosing the appropriate commands from the synthesis command menu, which controls both, the match matrix and grid diagram. To make a match, the user selects MakeMatch command from the menu. The system, in response, determines the most constrained matches for the present state. If only one such match is found, then it is selected as the next match. If more than one most constrained matches are found, then the system pops up a second level menu containing a list of these matches and the user selects the desired match from this list by clicking the left or middle button of the mouse. If the match is feasible only at one end, the match is made at that end; otherwise, the system asks the user to specify the end by popping up the third level menu with two options, hot and cold. As a consequence of making this match, the match matrix and the grid diagram are updated as described in Figure 4-2 and section 4.2.2. The system is now ready to execute the next synthesis command. If the system does not find any most constrained match, then a check is made to see if the current state is a solution state or a dead-end state. It informs the user of its finding by displaying appropriate message in the prompt window. If a dead-end has been reached, the system advises the user to backtrack using the UNMATCH operator. To undo the last match, the user selects Unmatch command. The match created by the preceeding MakeMatch command is removed and the match matrix and the grid diagram are restored accordingly (cf. Figure 4-3 and section 4.2.2).
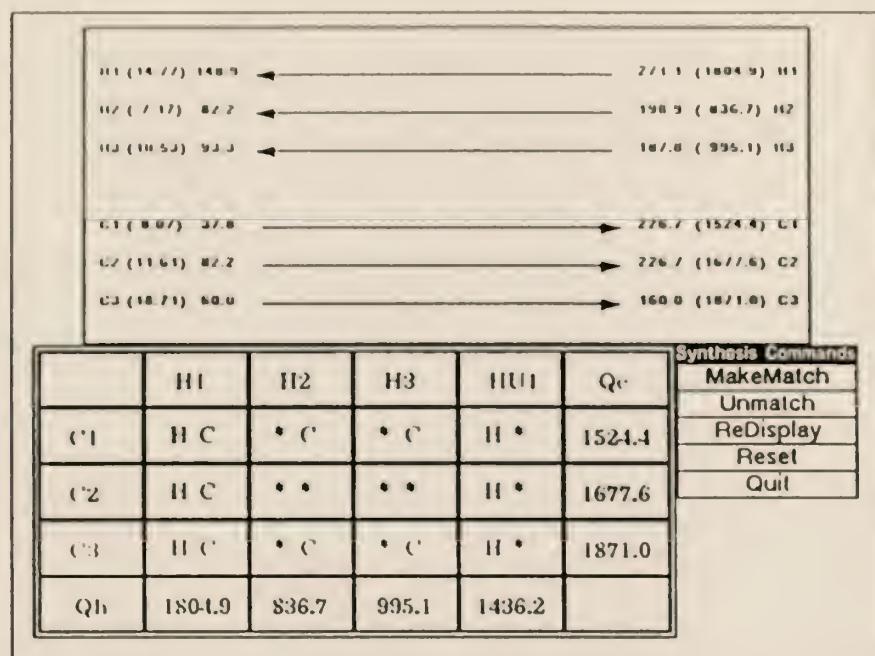
Fig. 4-8. Screen image at the end of initialization
of 6SP2 problem.

On initialization, the 6SP2 problem is in start state, S, as shown in Figure 4-9. Next, we will see how the proposed control strategy synthesizes a network for this problem.

(1) The match matrix of Figure 4-9 reveals that H2 and H3 are the most constrained streams; each can be matched in two possible ways. Furthermore, each stream can be matched with the same two cold streams C1 and C3. Thus, in this state, there are four most constrained matches: H2/C1, H3/C1, H3/C3 and H2/C3. On selection of MakeMatch command (MATCH operator), the system pops-up a menu with these four choices. In keeping with the proposed control strategy, H2/C1 is chosen at random; making this match will transform the problem into state 1, as shown in Figure 4-10. The resulting HTU, a heat exchanger, has a heat load of 836.7 units, and it eliminates the hot stream H2.

(2) As can be seen from Figure 4-10, in state 1 the problem has only one most constrained match, H3/C3. It is the only match possible for H3. The selection of MakeMatch command, therefore, makes this match without the user's intervention. This match transforms the problem to state 2, shown in Figure 4-11. The resulting HTU, a heat exchanger, has a heat load of 995.1 units. Hot stream H3 gets eliminated as a result of this match.

(3) In state 2 (Figure 4-11), cold stream C1 is the most constrained stream with only two possible matches. Out of these two matches, H1/C1 and HU1/C1, the second one is more constrained than the first one; hot stream H1 can be matched in five different ways, whereas hot utility HU1 can be matched in three ways. Therefore,

89

```
H1 (14.77) 148.9  ◄──────────────────────  271.1   (1804.9) H1

H2 ( 7.17)  82.2  ◄──────────────────────  198.9   ( 836.7) H2

H3 (10.53)  93.3  ◄──────────────────────  187.8   ( 995.1) H3


C1 ( 8.07)  37.8  ──────────────────────►  226.7   (1524.4) C1

C2 (11.61)  82.2  ──────────────────────►  226.7   (1677.6) C2

C3 (18.71)  60.0  ──────────────────────►  160.0   (1870.9) C3
```
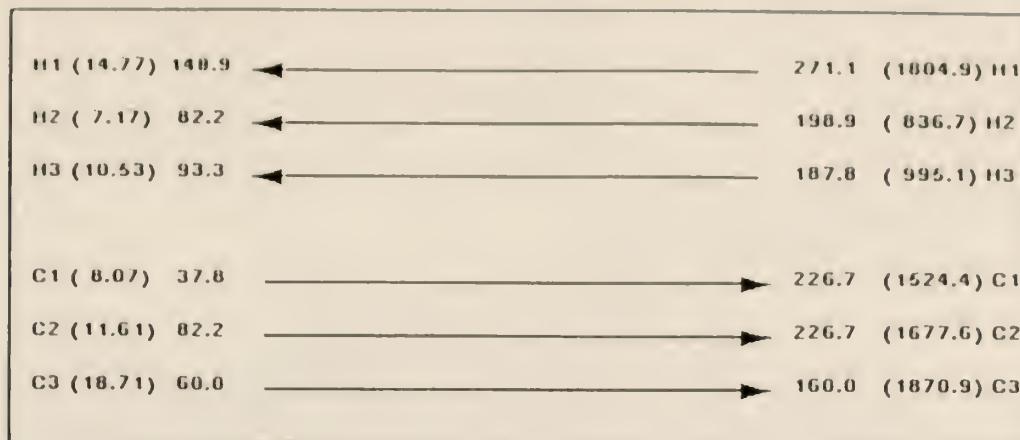
	H1	H2	H3	HU1	Qc
C1	H C	* C	* C	H *	1524.4
C2	H C	* *	* *	H *	1677.6
C3	H C	* *	* C	H *	1870.9
Qh	1804.9	S36.7	995.1	1436.2	

Fig. 4-9. 6SP2 problem: starte state.

90

```
H1 (14.77) 148.9  ◄─────────────────────────────  271.1  (1804.9) H1

H2 ( 7.17) 198.9  ◄──────①──────────────────────  198.9  (   0.0) H2

H3 (10.53)  93.3  ◄───────┼──────────────────────  187.8  ( 995.1) H3


C1 ( 8.07) 141.5  ─────────○─────────────────────►  226.7  ( 687.7) C1
                        ( 836.7)
C2 (11.61)  82.2  ───────────────────────────────►  226.7  (1677.6) C2

C3 (18.71)  60.0  ───────────────────────────────►  160.0  (1870.9) C3
```
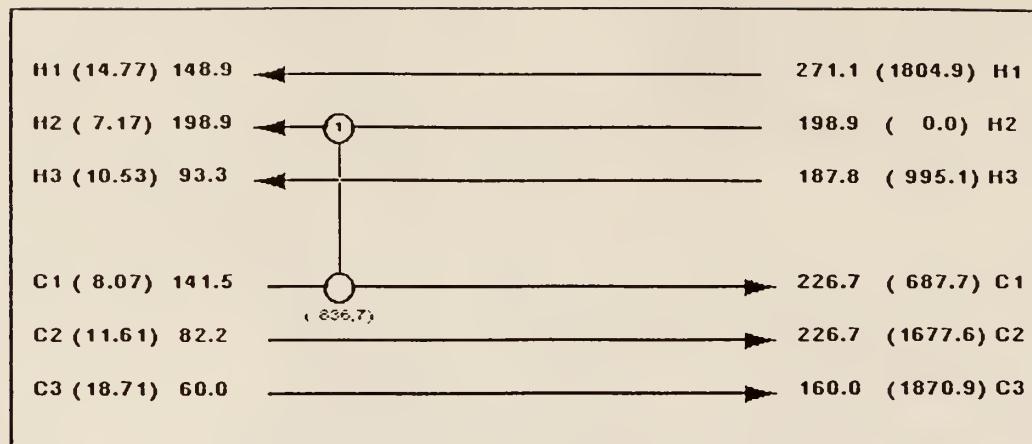
	H1	H2	H3	HU1	Qc
C1	H *	836.7	* *	H *	687.7
C2	H C	-----	* *	H *	1677.6
C3	H C	-----	* C	H *	1870.9
Qh	1804.9	0.0	995.1	1436.2	

Fig. 4-10. 6SP2 problem: state 1.

91

Fig. 4-11. 6SP2 problem: state 2.
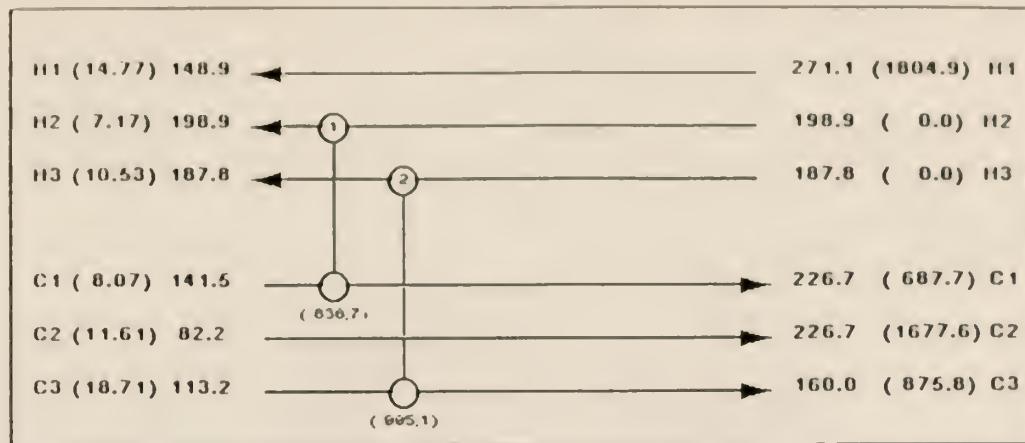
	H1	H2	H3	HU1	Qc
C1	H *	836.7	-----	H *	687.7
C2	H C	-----	-----	H *	1677.6
C3	H C	-----	995.1	H *	875.8
Qh	1804.9	0.0	0.0	1436.2	

match HU1/C1 is made as the response of the next MakeMatch
command. Making this match transforms the problem into state 3,
in Figure 4-12. The resulting HTU, a heater, has a heat load of
687.7 units and it eliminates cold stream C1.

(4) State 3 (Figure 4-12) has hot utility HU1 as the most constrained
stream (two matching possibilities). Both matches, HU1/C2 and
HU1/C3, are equally constrained since the cold streams C2 and C3
are identically constrained (three matching possibilities for
each). Thus, both the matches (HU1/C2 and HU1/C3) qualify as the
most constrained one. On selection of MakeMatch command,
therefore, the system will ask the user to choose one from a
pop-up menu. Match HU1/C3 is chosen at random. This match
transforms the problem into state 4, Figure 4-13. The resulting
HTU, again a heater, has a heat load of 748.5 units. It
eliminates hot utility HU1.

(5) In state 4 (Figure 4-13), the most constrained streams are C2 and
C3 (two matching possibilities apiece). Since both the streams
match with the only remaining hot stream H1 at hot as well as
cold end, both the matches, H1/C2 and H1/C3, qualify for the most
constrained match. Once again, based on random selection, H1/C3
match is chosen from the menu popped-up by the system. The match
is feasible at both ends; so, once again through the menu hot end
is chosen at random. Making H1/C3 match at hot end transforms
the problem into state 5, as shown in Figure 4-14. This match, a
heat exchanger, has a heat load of 127.4 units and it eliminates
cold stream C3.
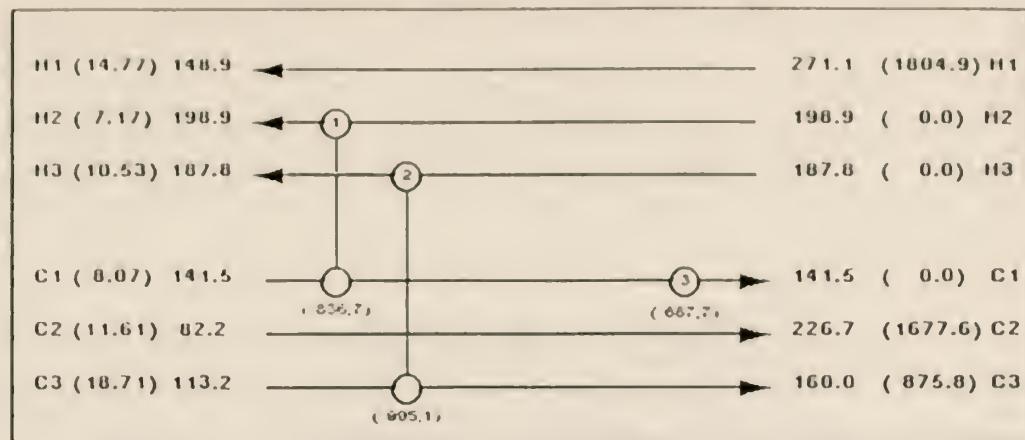
	H1	H2	H3	HU1	Qc
C1	-----	836.7	-----	687.7	0.0
C2	H C	-----	-----	H *	1677.6
C3	H C	-----	995.1	H *	875.8
Qh	1804.9	0.0	0.0	748.5	

Fig. 4-12. 6SP2 problem: state 3.
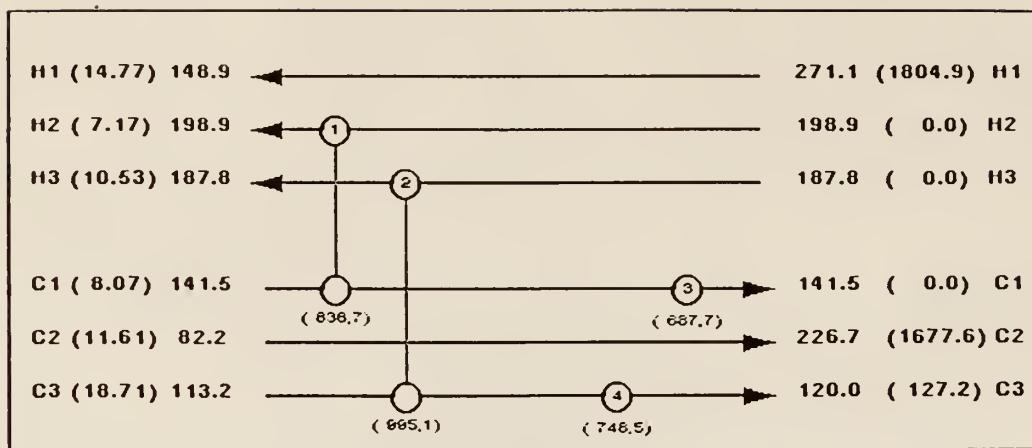
	H1	H2	H3	HU1	Qc
C1	-----	836.7	-----	687.7	0.0
C2	H C	-----	-----	-----	1677.6
C3	H C	-----	995.1	748.5	127.2
Qh	1804.9	0.0	0.0	0.0	

Fig. 4-13. 6SP2 problem: state 4.
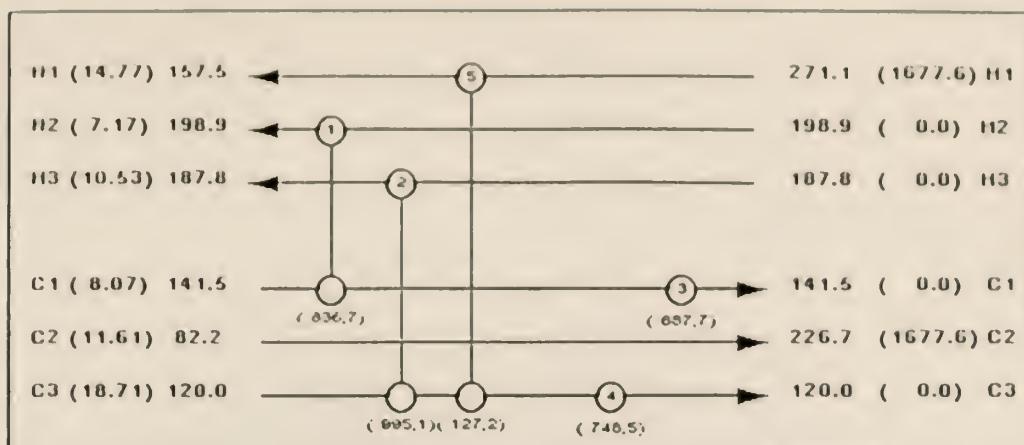
	H1	H2	H3	HU1	Qc
C1	-----	836.7	-----	687.7	0.0
C2	H C	-----	-----	-----	1677.6
C3	127.2	-----	995.1	748.5	0.0
Qh	1677.6	0.0	0.0	0.0	

Fig. 4-14. 6SP2 problem: state 5.

(6) As revealed by Figure 4-14, state 5 has only one possible match (the last one), H1/C2. It is feasible at both, hot and cold ends. However, for this match, both ends are equivalent, since both the streams are getting eliminated. Making this last match transforms the problem into state 6, which, as shown by Figure 4-15, is a goal state. The resulting HTU, a heat exchanger with a heat load of 1677.5 units, eliminates the last two streams, hot stram H1 and col stream C2.

The search tree explicitly generated in the process of arriving at the present solution is shown in Figure 4-16. Each state is represented by a node (circle) with appropriate label. Each arc (connecting a pair of circles) represents the application of MATCH operator, with the selected match ID as its label. Nodes labeled 1 through 6 are the states described in the preceeding paragraphs, whereas those labeled a through g are the states reached by choosing the alternate most-constrained-matches in steps (1), (4) and (5) above.

On continuing the search by backtracking with the help of alternate choices available in steps (1), (4) and (5) above, the search graph shown in Figure 4-17 is generated. Each node in this search graph (a circle in the diagram) represents a problem state and each arc (a line connecting two nodes), an application of MATCH (or UNMATCH while backtracking) operator. The start state is indicated by an "S" in the corresponding node. Each arc is labelled with the match selected for the operator. In case of a match feasible at both ends, the corresponding labels are qualified with a [c] or an [h] corresponding to the cold and hot end match, respectively. A solution state as well as a

97

H1 (14.77) 157.5   157.5 (0.0)) H1

H2 (7.17) 198.9   198.9 (0.0) H2

H3 (10.53) 187.8   187.8 (0.0) H3

C1 (8.07) 141.5   141.5 (0.0) C1

C2 (11.61) 82.2   82.2 (0.0)) C2

C3 (18.71) 120.0   120.0 (0.0) C3

(836.7)   (687.7)

(1677.6)

(995.1)(127.2)   (748.5)
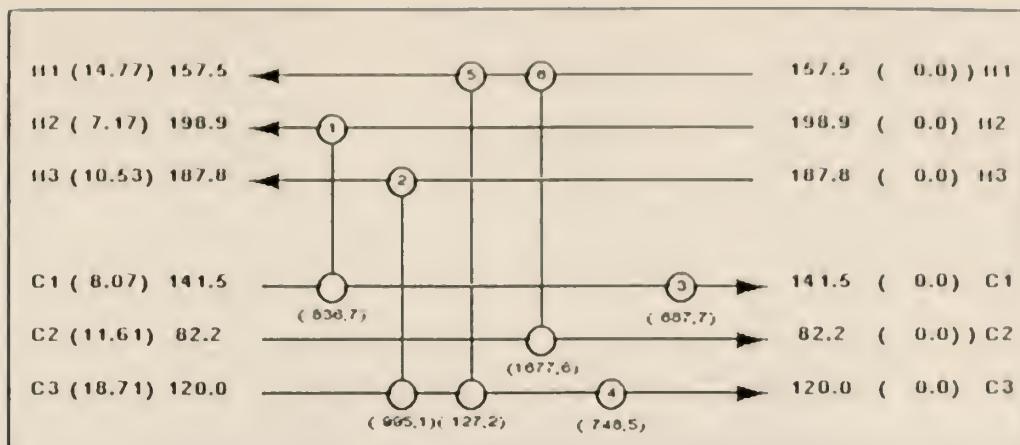
	H1	H2	H3	HU1	Qc
C1	-----	836.7	-----	687.7	0.0
C2	1677.6	-----	-----	-----	0.0
C3	127.2	-----	995.1	748.5	0.0
Qh	0.0	0.0	0.0	0.0	

Fig. 4-15. 6SP2 problem: state 6.

98
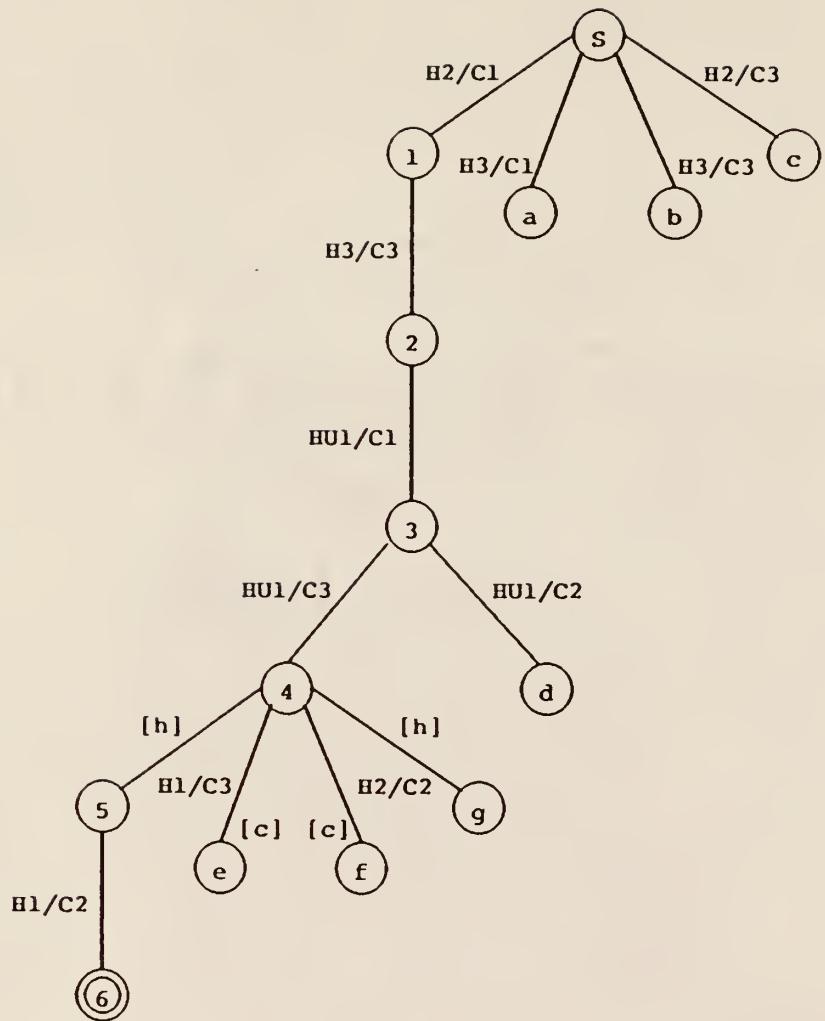
Fig. 4-16. 6SP2 problem: search tree for the first solution.
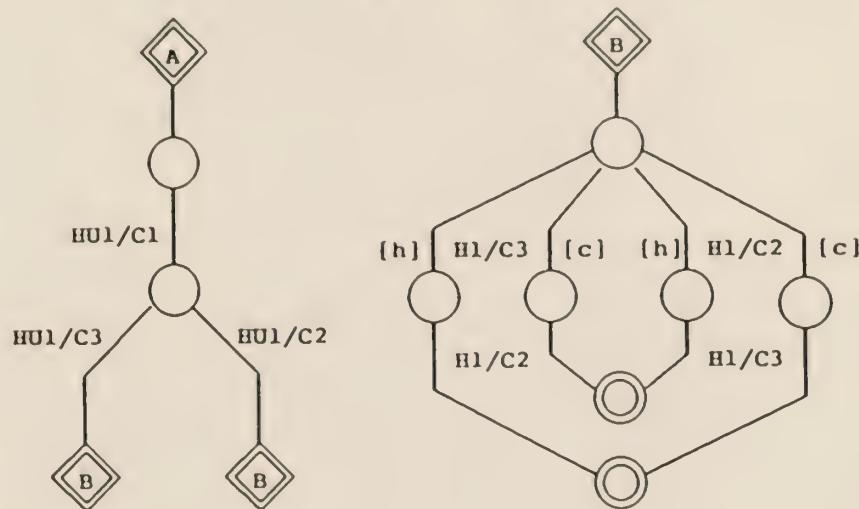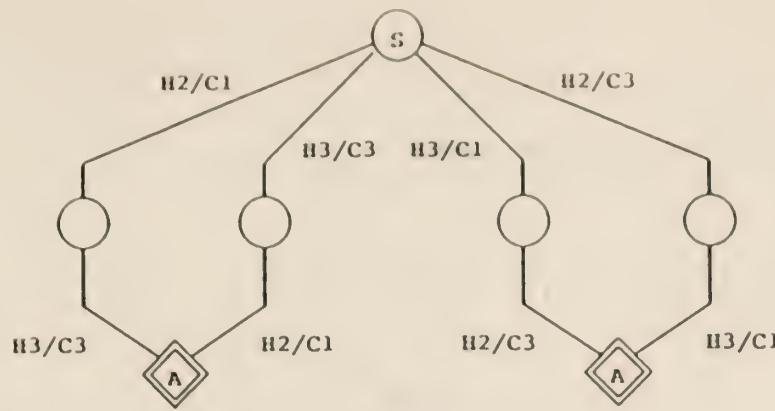
Fig. 4-17.   Search graph for 6SP2 problem.

100

dead-end state in the graph is indicated by a smaller concentric circle in the corresponding node. To differentiate the two, the inner circle is shaded for a dead-end state, whereas for a solution state, it is not. For the ease and convenience of drawing and understanding the graph, it is broken up into several substructures, labelled A, B, C, etc., in diamond-shaped boxes. A substructure appears one or more times in the top level graph and/or other substructures.

As can be seen from the search graph in Figure 4-17, the system has generated eight distinct networks without encountering a single dead-end.

4.3.2 Evaluation of Control Strategy

As seen in the previous section, the system has provided satisfactory results for the 6SP2 problem. The system has been tested with three additional problems in a manner similar to that for the 6SP2 problem. In each case, only the search graph explicitly generated by the system is reported and discussed here. The individual network configurations are not included here.

The data for 7SP2 test problem [Masso and Rudd, 1969] is shown in Figure 4-18. The problem consists of three hot streams, H1, H2, and four cold streams, C1, C2, C3, C4. The minimum utility requirement for the problem is one hot utility stream, HU1, with a heat duty of 217.6 units. The minimum driving force is 20 degrees. The search graph explicitly generated for this problem is shown in Figure 4-19. As can be seen from this graph, once again, the control strategy succeeds in

101

Stream ID	mc_p	Source Temp	Target Temp
H1	2.376	590	400
H2	1.577	471	200
H3	1.32	533	150
C1	1.60	200	400
C2	1.60	100	430
C3	4.128	300	400
C4	2.624	150	280

$\Delta T_{min} = 20°$

Minimum Utility Requirement: hot utility (HU1): 217.6 units.

Fig. 4-18. Data for 7SP2 problem.
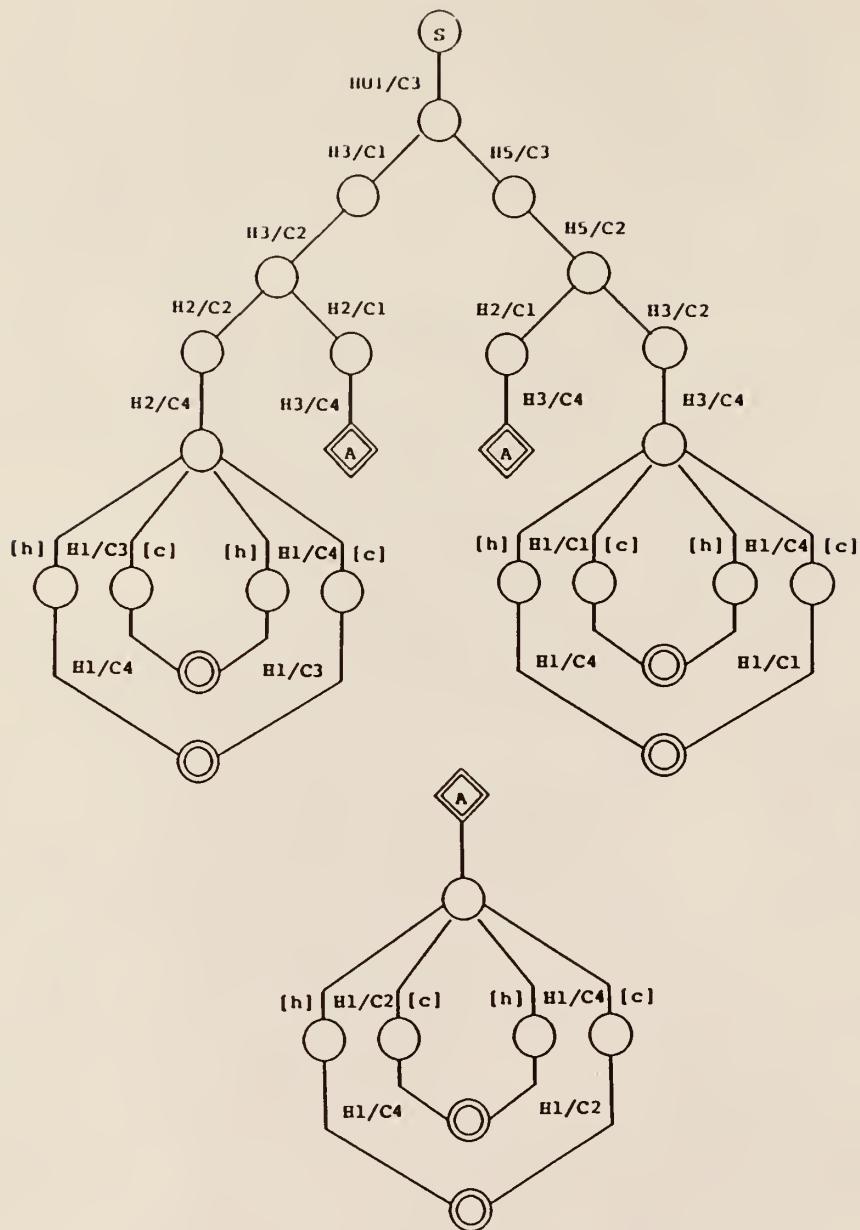
Fig. 4-19.   Search graph for 7SP2 problem.

avoiding all dead ends while generating eight distinct network configurations.

Figure 4-20 shows the data for the third test problem, the 10SP1 problem [Pho and Lapidus, 1973]. The problem consists of five hot process streams, H1 through H5, and five cold process streams, C1 through C5. The minimum utility requirement consists of one cold utility, CU1, with a heat duty of 1877 units. The minimum driving force is 11.1 degrees. Figure 4-21 displays the search graph for this problem. The system generated twenty distinct network configurations, but in the process, encounters six dead-ends. All the dead-ends are confined to one substructure of the graph, labeled B. The match responsible for all the dead-ends, viz., H5/C4 match at the cold end, could not be avoided by the present control strategy.

The fourth and the last test problem is the 7SP1 problem [Masso and Rudd, 1969], the data for which is shown in Figure 4-22. The problem consists of three hot process streams, H1 through H3, and four cold streams, C1 through C4. The minimum utility requirement is one cold utility, CU1, with a heat duty of 1203.2 units. The search graph for this problem is displayed in Figure 4-23. In sharp contrast to its performance for the previous test problems, the system fails to generate a single network configuration and encounters eight dead-ends. Note that the system can obtain split candidate networks (in which one or more streams are split), if operators SPLIT and MERGE are available. The present results indicate that the system has failed to obtain an unsplit network configuration, which is usually preferred over a split configuration, even if both feature the minimum number of HTUs and the

104

Stream ID	mc_p	Source Temp	Target Temp
H1	8.79	160.0	93.3
H2	10.55	248.9	137.8
H3	14.77	226.7	65.6
H4	12.55	271.1	148.9
H5	17.72	198.9	65.6
C1	7.62	60.0	160.0
C2	6.08	115.6	221.7
C3	8.44	37.8	221.1
C4	17.28	82.2	176.7
C5	13.90	93.3	204.4

$\Delta T_{min} = 11.1°$

Minimum Utility Requirement: cold utility (CU1): 1877 units.

Fig. 4-20. Data for 10SP1 problem.
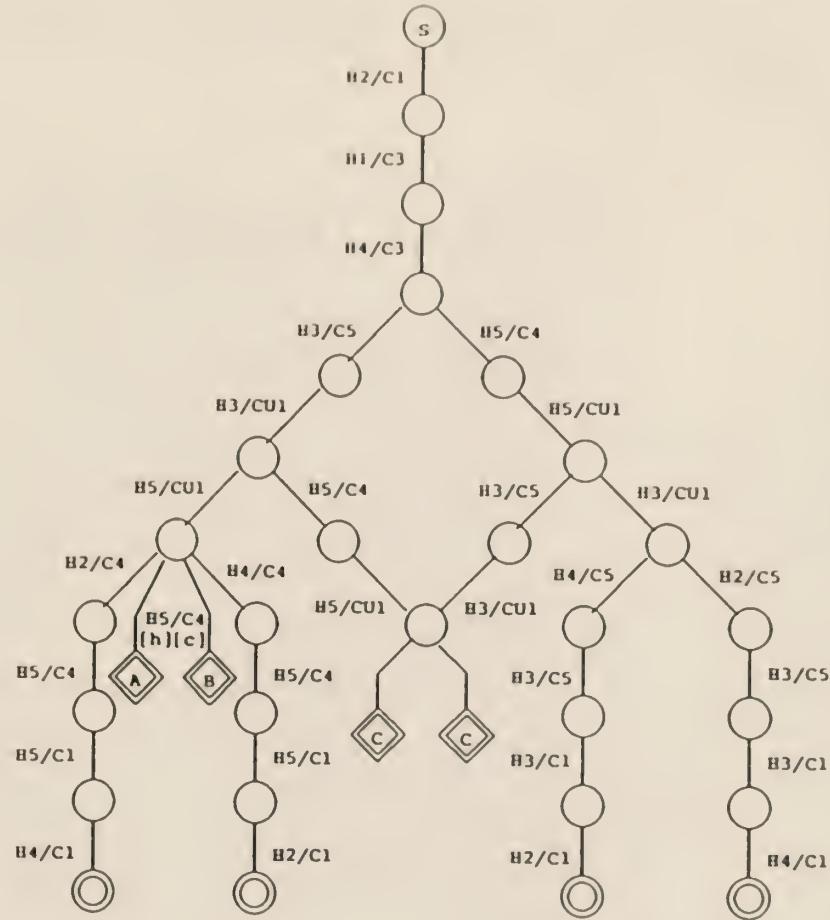
Fig. 4-21.   Search graph for 10SP1 problem (contd.).

106a

Fig. 4-21. Search graph for 10SP1 problem (contd.).

106b
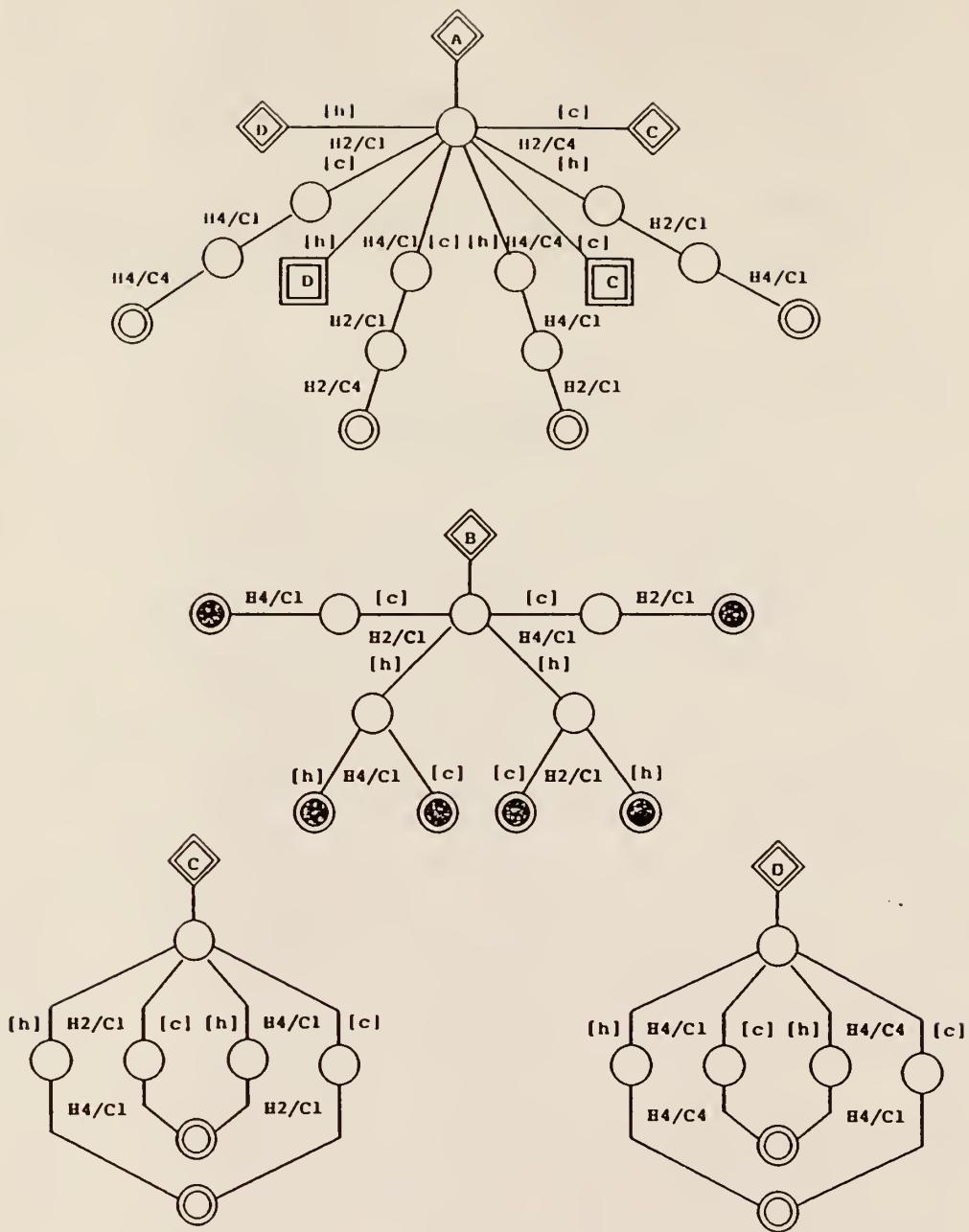
Stream ID	mc_p	Source Temp	Target Temp
H1	14.77	226.7	65.6
H2	12.56	271.1	148.9
H3	17.72	198.9	65.6
C1	8.44	37.8	221.1
C2	17.28	82.2	176.7
C3	13.90	93.3	104.4
C4	10.47	176.7	210.0

$\Delta T_{min} = 11.1°$

Minimum Utility Requirement: cold utility (CU1): 1203.2 units.

Fig. 4-22. Data for 7SP1 problem.
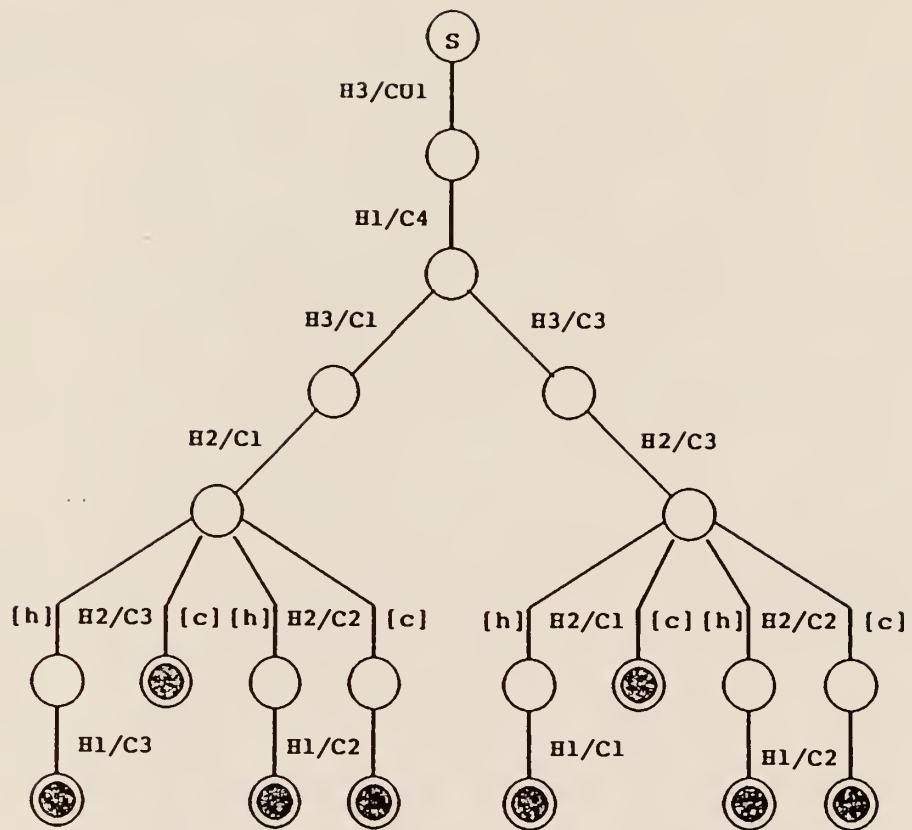
Fig. 4-23. Search graph for 7SP1 problem.

minimum utility consumption. The inability of the system to obtain an unsplit network is due to the fact that the present control strategy makes decisions based on the "current" feasibilities of the matches. It does not take into account the "future" changes in the feasibilities of other matches, as the consequence of a match being made "at present". Making a match between a pair of streams changes the feasibilities of some or all of the matches that involve the remaining or uneliminated stream. The number of such matches that change their feasibilities on making a match determines how constrained a problem is. For a highly constrained problem, the control strategy does not perform well (e.g., 7SP1 problem); for a less constrained problem, it performs excellently (e.g., 6SP2, 7SP2 and 10SP1 problems). The possible directions for future enhancement of the control strategy to alleviate this situation are suggested in chapter 5.

CHAPTER 5.   CONCLUSIONS AND RECOMMENDATIONS

In the present work, an AI based approach has been introduced for automating the synthesis (preliminary design) of energy integration networks. The approach relies on the explicit usage of domain knowledge to reduce the complexity associated with the search for desired solution(s). All prevailing approaches to this design task attempt to find a network configuration with the minimum annual cost ($/year). However, since this network usually possesses undesirable operational characteristics, alternate configurations having near-minimum cost, are required for the detailed design phase. No existing computer based synthesis method is suitable for this purpose.

Good designers usually employ qualitative relationships between the cost of a network and its structural characteristics, such as the number of units (HTUs) and the amount of utility consumption, for generating the alternate network configurations required for the detailed design phase. The AI based approach adopted in the present work utilizes these relationships and focuses on the structural characteristics of the network being generated, rather than on the annual cost. In other words, the problem of finding a set of networks, each with a near-minimum cost, has been transformed into that of finding a set of structures possessing specific structural characteristics, viz., the minimum number of units and the minimum utility consumption.

The problem of finding candidate networks with the desired structural characteristics has been formulated as a state-space search problem in the present work. The state-space, its representation, and

110

four operators for state transformations have been defined. The most significant aspect of this formulation is that it utilizes the available domain knowledge, in the form of a set of feasibility rules based on the elimination strategy and the associated necessary and sufficient conditions, to reduce the extent of search required to obtain the desired solutions. A control strategy has been proposed to exploit the domain knowledge for minimizing the backtracking during the search.

To demonstrate the feasibility and effectiveness of the proposed AI based approach as well as to evaluate the proposed control strategy, the search system formulated in the present work has been implemented on a Xerox AI workstation using the knowledge programming environment LOOPS. Of the four different programming paradigms offered by the LOOPS environment, the present prototype, termed HENSYN, has been built entirely within the object-oriented paradigm. The structure of the prototype has been described with the help of the implementational details of the user interface and the three components of the search system (the state space, operators and control strategy). Two operators, MATCH and UNMATCH, out of the four required, have been implemented in the present prototype. It is possible to solve a number of problems without the remaining two operators, SPLIT and MERGE. The usage of the HENSYN system has been demonstrated in detail with the help of a test problem (the so-called 6SP2 test problem). Moreover, the prototype system has been tested by solving three additional synthesis problems taken from the literature (the so-called 7SP1, 7SP2 and 10SP1 test problems). A performance analysis has been presented with the results of the four test problems. The HENSYN system has found eight

111

distinct network configurations each for the 6SP2 and 7SP2 problems, twenty configurations for the 10SP1 problem and none for the 7SP1 problem. In the first two problems, all dead-ends were successfully avoided, whereas for the last two problems, six and eight dead-ends, respectively, were encountered. The system requires the remaining two operators SPLIT and MERGE to obtain one or more solutions the 7SP1 problem, though unsplit solutions exist. This inability of the present prototype to obtain the unsplit network configurations has been analyzed and its cause has been identified. The possible remedy is suggested in a later paragraph.

The capabilities of the HENSYN system can be summarized as follows.

(a)   It is capable of generating multiple network configurations without searching the state space multiple times. Each of these networks has a near-minimum cost, or equivalently, they feature the minimum number of HTUs and the minimum utility consumption. This capability is due solely to the knowledge it possesses about the feasibilities of all matches.

(b)   It has an efficient control strategy that utilizes the domain knowledge to minimize the backtracking during the search, by successfully avoiding most of the dead-ends.

(c)   It provides an effective visualization of problem states as well as the partial and complete networks.

(d)   It has a menu-driven, user-friendly interface that enables a user to synthesize HENs easily and efficiently.

(e)   The system has excellent modularity. It is easy to modify any parts of the system without excessive "follow-up" changes. This

112

power is derived from the object-oriented paradigm of LOOPS, upon which it has been built.

(f) It separates the domain knowledge and the control mechanism that uses this knowledge. Thus, addition or modification in any one will not affect the other.

Based on the results of the present work, we can conclude that the AI based approach is not only feasible and useful, but also superior to the conventional approaches for automating the conceptual design (synthesis) tasks. It reduces the complexity of a design task by appropriately using the available domain knowledge. The approach, coupled with the knowledge programming tools and techniques, enables us to develop incrementally an "intelligent" computer-aided design system by updating its knowledge content and/or enhancing the search techniques (control strategies). Finally, a useful side benefit of developing such a system is that the necessary design knowledge, not readily available through the textbooks or classroom instruction, gets formalized. Consequently, the approach employed in the present work is recommended for developing the automated problem solving system in a variety of engineering design domains, such as the electrical power distribution systems, digital circuits, chemical process flowsheets, waste water and sewage treatment, and piping layouts for water distribution systems. Additionally, the approach can also be recommended for solving problems involving the resource allocation and scheduling with constraints in other domains, such as distributed computer systems, resource management for large scale computer systems, financial planning and project management.

The present work can be extended in two ways. The first is the improvement of the quality of the networks generated by the system. It involves extraction and formalization of additional domain knowledge, relating other structural characteristics (e.g., the average driving force in each HTU and the distribution of heat loads of the HTUs) to the annual cost as well to the other operational characteristics, such as controllability, resiliency, and flexibility. This task requires extensive participation of domain experts. The present prototype can aid in this task by acting as an experimentation device for the participating domain experts and knowledge engineers. The second form of extension is the enhancement of the performance (efficiency) of the system in obtaining the candidate networks. It deals with identifying and remedying the limitations of the present prototype to improve its efficiency. Towards this end, the following two enhancements are suggested.

(a) The performance of the system is dependent on the nature of the problem. For highly constrained problems, such as 7SP1, it does not yield good results, in that, unsplit network configurations can not be found. This limitation can be overcome by imparting a "look ahead" capability to the control strategy. This would involve a thorough analysis of the effects of the characteristic values of a pair of streams on the changes in the feasibilities of their match.

(b) The system currently supports only two operators, MATCH and UNMATCH, out of the four required for HEN synthesis. The remaining two, SPLIT and MERGE, are required for solving some

problems (not all problems require stream splitting). These operators can be implemented as sequences of methods in the same fashion as the two operators implemented in the present system. Nevertheless, it should be noted that the present system can be utilized for synthesizing HENs requiring stream splitting, provided that the user specifies each substream as a separate stream. Such an approach is not elegant, but is workable.

With these enhancements, the HENSYN system can be employed as an automated synthesis system for the design of energy integration networks. Even without these enhancements, i.e., in the present form, the prototype can be used by novices (students and inexperienced designers), since it performs better than them; however, an experienced designer may be able to outperform the present prototype. In such cases, the system is still useful as a design tool; the user can override the system's decisions whenever required.

REFERENCES

1. Barr, A., and Feigenbaum, E. A., *The Handbook of Artificial Intelligence, vol 1*, William Kaufmann, Inc., Los Altos, CA (1981).

2. Barton, I., Jones, D. H. and Smith, G. J., "New Developments in Heat Exchanger Network Targeting, Design and Analysis," paper presented at the AIChE Summer Meeting, Minneapolis, August 19 (1987).

3. Bobrow, D. G. and Stefik, M., *The LOOPS Manual*, Xerox PARC (1983).

4. Boland, D., and Linhoff, B., "The Preliminary Design of Networks for Heat Exchange by Systematic Methods," *The Chemical Engineer*, 222-228, April (1979).

5. Cannon, H. I., "Flavors: A non-hierarchical approach to object-oriented programming," in *LOOPS Manual*, Xerox PARC (1982).

6. Cena, V., Mustacci, C. and Natali,F., "Synthesis of Heat Exchanger Networks: A non-iterative Approach," *Chem. Eng. Sci.*, 32, 1227 (1977).

7. Cerda, J. and Westerberg, "Minimum Utility Usage in Heat Exchanger Network Synthesis - A Transportation Problem," DRC Report No. 06-16-80, Carnegie-Mellon University, Pittsburgh, PA (1980).

8. De Kleer, J., "How Circuits Work," in *Qualitative Reasoning about Physical Systems*, (Ed.) D. G. Bobrow, The MIT Press, Cambridge, MA, 205-280 (1985).

9. Fikes, R. and Kehler, T., "The Role of Frame-Based Representation in Reasoning," CACM 28 (9), pp. 904-920 (1985).

10. Goldberg, A., "Introducing the Smalltalk-80 System," *Byte*, 6 (8), August (1981).

11. Goldberg, A. and Robson, D., *Smalltalk-80: The language and Its Implementation*, Addison-Wesley, Reading, MA (1983).

12. Greenkorn, R. A., Koppel, L. B., and Raghawan S., "Heat Exchanger Network Synthesis - A Thermodynamic Approach," paper presented at the 71st AIChE Meeting, Miami, FL (1978).

13. Grossmann, I. E., and Sargent, R. W. H., "Optimum Design of Heat Exchanger Networks," *Comp. & Chem. Eng.*, 2 (1), (1978).

14. Hohmann, E. C., "Optimal Networks for Heat Exchange," Ph.D. Thesis, Univ. S. Calif. (1971).

15. Ingalls, D. H., "The Smalltalk-76 Programming System: Design and Implementation," in *Conference Record of the Fifth Annual ACM*

117

Symposium Principles of Programming Languages, Tuscon, AZ, pp. 9-16, January (1978).

16. Jezowsky, J. and Hahne, E., "Heat Exchanger Network Synthesis by a Depth-First Method -- A Case Study," *Chem. Eng. Sci.*, 41 (12), 2989-2997 (1986).

17. Kesler, M. G. and Parker, R. O., "Optimal Networks of Heat Exchanger," Chem. Eng. Progr. Sym. Series, No. 92, 61, 111 (1969).

18. Kobayashi, S., Umeda, T. and Ichikawa, A., "Synthesis of Optimal Heat Exchange Systems -- An Approach by the Optimal Assignment in Linear Programming," *Chem. Eng. Sci.*, 26, 3176 (1971).

19. Kyle, B. G., *Chemical and Process Thermodynamics*, Prentice-Hall, Englewood Cliffs, NJ, p. 22 (1983).

20. Linhoff, B., and Flower, J. R., "Synthesis of Heat Exchanger Networks, Part I. Systematic Generation of Energy Optimal Networks," AIChE J., 24, 633 (1978).

21. Linhoff, B. and Hindmarsh, E.; "The Pinch Design Method of Heat Exchanger Networks," (1983).

22. Linhoff, B., Townsend, D. W., Boland, D., Hewitt, G. F., Thomas, B. E. A., Guy, A. R. and Marsland, R. H., *A User Guide on Process*

Integration for the Efficient Use of Energy, The Institute of Chemical Engineers, Rugby, England (1982).

23.   Masso, A. H. and Rudd, D. F., "The Synthesis of System Designs. II. Heuristic Structuring," *AIChE J.*, <u>15</u>, 10 (1969).

24.   Mehta, C. D., "Knowledge Based Approach for Process Synthesis Automation: An Application to Heat Exchanger Network Synthesis," M. S. Thesis, Kansas State University, Manhattan, KS (1986).

25.   Nilsson, N. J., *Principles of Artificial Intelligence*, Tiago, Palo Alto, CA (1980).

26.   Nishida, N., Stephanopoulos, G., and Westerberg, A. W., "A Review of Process Synthesis," *AIChE J.*, <u>27</u>, 321-351 (1981).

27.   Papoulias, S. A. and Grossmann, I. E., "A Structural Optimization Approach in Process Synthesis - II. Heat Recovary Networks," *Comp. & chem. Eng.*, <u>7</u> (6), 707-721 (1983).

28.   Pho, T. K. and Lapidus, L., "Topics in Computer-Aided Design II. Synthesis of Optimal Heat Exchanger Networks by Tree search Algorithms," *AIChE J.*, <u>19</u>, 1182 (1973).

29. Rathore, R. N. S., and G. J. Powers, "A Forward Branching Scheme for the Synthesis of Energy Recovery Systems," *Ind. Eng. Chem. Process Design & Development*, 14, 175 (1975).

30. Rich, E., *Artificial Intelligence*, McGraw-Hill, New York (1983).

31. Shah, J. V. and Westerberg, A. W., "Evolutionary Synthesis of Heat Exchanger Networks," paper presented at the AIChE Annual Meeting, Los Angeles, CA (1975).

32. Stefik, M. and Bobrow, D. G., "Object Oriented Programming: Themes and Variations," *AI Magazine*, 6 (4), 40-62 (1986).

33. Teitelman, W., et al., *The Interlisp Reference Manual*, Xerox PARC, October (1978).

34. Williams, B. C., "Qualitative Analysis of MOS Circuits," in *Qualitative Reasoning about Physical Systems*, (Ed.) D. G. Bobrow, The MIT Press, Cambridge, MA, pp. 281-346 (1985).

35. Weinreb, D. and Moon, D., *Lisp Machine Manual*, Symbolics Inc, (1981).

36. Xerox, Interlisp Reference Manual, Xerox Palo Alto Research Center, October (1983).

KNOWLEDGE PROGRAMMING FOR COMPUTER-AIDED DESIGN:

AN APPLICATION TO THE DESIGN OF ENERGY INTEGRATION NETWORKS

by

CHETAN   MEHTA

B. Tech., Indian Institute of Technology, Bombay, 1982.
M. S., Kansas State University, 1986.

AN ABSTRACT OF A THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

ABSTRACT

An AI based approach is introduced in the present work for automating the preliminary or conceptual phase of design. The approach is used for a typical engineering design problem, viz., that of synthesizing energy integration networks for chemical and power plants. This synthesis problem is concerned with generating a set of networks, possessing acceptable lower costs (in $/year), for subsequent analysis and evaluation in the detailed design phase. By using the available qualitative relationships between the cost of a network and its structural characteristics, the problem of generating a set of near-minimum-cost networks is transformed into that of finding network configurations featuring the desired structural characteristics. The need and rationale for such a transformation are provided.

The problem of generating a set of energy integration networks, each featuring the desired structural characteristics, is formulated as a state-space search problem. The most distinguishing aspect of this formulation is that it focuses the search for a desired solution by utilizing the available domain knowledge about how to attain the desired structural characteristics for an energy integration network. A search system has been defined by identifying a scheme for representing the problem states and four operators for state transformations. A control strategy, which exploits the domain knowledge to minimize the extent of backtracking, is proposed. The search system generates multiple network configurations for a given problem without searching the state space multiple times.

To demonstrate the feasibility and effectiveness of the AI based approach, the search system defined in the present work has been implemented in the object-oriented environment LOOPS on a Xerox AI workstation. The structure of the prototype, viz., the implementational details of the three components of the search space and the user interface, is described. The usage of the system is exemplified by solving a test problem taken from the literature. The system has been tested by solving three additional problems. The performance analysis and evaluation of the prototype are discussed with the help of the results of the four test problems. Finally, the capabilities and limitations of the prototype are summarized and future enhancements of the prototype are proposed.